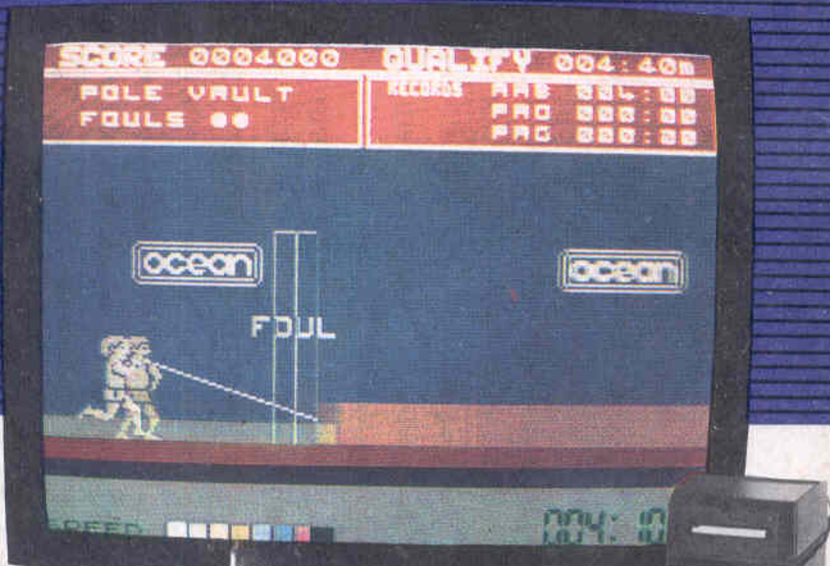
K. KURYŁOWICZ D. MADEJ K. MARASEK

PRZEWODNIK po ZX SPECTRUM



WYDAWNICTWA KOMUNIKACJI I ŁĄCZNOŚCI

PRZEWODNIK po ZX SPECTRUM

Naszym dziewczynom
dużym i małym

PRZEWODNIK po

Pragniemy podziękować:

— naszym rodzinom, a szczególnie żonom: *Ewie*, *Stasi* i *Kasi* za to, że znosiły cierpliwie niekończące się rozmowy o komputerach,

— naszym szefom: *Zbigniewowi Mikurendzie* z Uniwersytetu Łódzkiego, *Jerzemu Motylewskiemu* i *Władysławowi Mikielowi* z IPPT PAN za życzliwość i cierpliwość,

— recenzentom: prof. dr. hab. inż. *Jerzemu Zielińskiemu* i mgr. *Lechowi Czarnemu* za trud włożony w opinię,

— naszym kolegom: *Jolancie Dymalskiej* z UŁ oraz *Januszowi Mika*, *Ryszardowi Gubrynowiczowi* z IPPT PAN, *Rafałowi Gutkowskiemu* z IPI PAN — za cenne uwagi i materiały,

— redaktorowi *Izie Mika* za zaangażowanie.

AUTORZY

Warszawa, kwiecień 1986 r.

ZX SPECTRUM

Krzysztof Kuryłowicz
Dariusz Madej
Krzysztof Marasek

WKŁ

Wydawnictwa Komunikacji i Łączności
Warszawa 1986

Okładkę i stronę tytułową projektował: *Tadeusz Pietrzyk*
Zdjęcia wykonali:
Grażyna Bartłomiejczyk, *Zdzisław Machnicki*
Opiniodawcy:
prof. dr hab. inż. *Jerzy Zieliński*
mgr *Lech Czarny*
Redaktor merytoryczny: mgr inż. *Izabela Ewa Mika*
Redaktor techniczny: *Alicja Jabłońska-Chodzeń*
Korekta: *Zespół*

Książka jest vademecum wiedzy o ZX Spectrum. Zawiera informacje o instalowaniu, użytkowaniu i budowie komputera. Uczy programowania w języku Basic ZX Spectrum, tworzenia grafiki i dźwięku. Omawia programowanie w języku asemblera oraz podaje przegląd języków programowania dostępnych na tym komputerze. Opisuje microdrive, układ Interface 1 oraz pamięci dyskowe. Zawiera również omówienie programów do nauki, pracy i zabawy.

ISBN 83-206-0691-8

Wydawnictwa Komunikacji i Łączności
Warszawa 1986
Wydanie I. Nakład 149 650+350 egz.
Ark. wyd. 19. Ark. druk. 18
Oddano do składania w maju 1986
Podpisano do druku w październiku 1986
Papier offset kl. V 70 g rola 84 cm
Zamówienie P/56/86. K.9777. P-33
Skład: Drukarnia im. Rewolucji Paźdź. W-wa
Druk i oprawa: Zakłady Graficzne w Gdańsku
Zam. druk. 2010/86
Cena zł 489+1 na NFOZ

Spis treści

Przewodnik po Przewodniku po ZX Spectrum
* dla początkujących
** dla hobbistów
*** dla zaawansowanych

Wstęp

Sir Clive Sinclair oraz zespół projektantów ZX Spectrum nie przypuszczali zapewne, że komputer ten zrobi w Polsce tak oszałamiającą karierę. Któż bowiem mógł przewidzieć, że znajdzie on zastosowanie do sterowania procesami technologicznymi, czy też będzie wykorzystywany do prowadzenia prac badawczych w instytutach naukowych. Prawdą jest jednak, że najczęściej jest używany do zabawy.

Szacuje się, że w Polsce znajduje się ponad 18 000 komputerów ZX Spectrum (połowa 1985 r.) i liczba ich wzrasta. Spotkać je można w różnych instytucjach, szkołach, wyższych uczelniach, klubach komputerowych. Jednak większość z nich znajduje się w rękach prywatnych.

Jednocześnie w Polsce używa się kilku tysięcy zachodnich programów przeznaczonych dla ZX Spectrum: gier, programów edukacyjnych, a także programów użytkowych: translatorów, edytorów tekstu itp.

Specjaliści (zresztą nie tylko oni) wskazują na wady tego komputera:

— zasadniczą wadą jest współpraca z magnetofonem (źle działający magnetofon może skutecznie zniechęcić do używania Spectrum),
— zbyt słaba (mechanicznie) jest konstrukcja komputera,
— klawiatura jest niewygodna i dosyć szybko psuje się,
— niezawodność jest mniejsza niż innych komputerów domowych,
— dosyć wolno pracuje oprogramowanie systemu (zawiera ono także błędy),
— brak jest zabezpieczeń przed uszkodzeniami mechanicznymi i elektrycznymi (np. przepięciami),
— brak jest wejść pozwalających bezpośrednio podłączyć manipulator (ang. *joystick*) oraz urządzenia zewnętrzne, pracujące w standardzie RS 232 lub CENTRONICS.

Spectrum jednak, jak na razie, nie musi obawiać się krytyki. Nie ma bowiem konkurenta. Jest najtańszym komputerem domowym. Ma przy tym:

— dosyć dużą pamięć (48 kbajtów RAM, 16 kbajtów ROM),
— niezły procesor (Z80A),
— kolorową grafikę wysokiej rozdzielczości,

— możliwość wytwarzania dźwięków,
— bogaty język Basic,
— możliwość współpracy z normalnym telewizorem,
— prostą konstrukcję,
— wszystkie sygnały procesora są wyprowadzone na zewnątrz (ułatwia to rozbudowę komputera),
— bogate, różnorodne oprogramowanie.

Spectrum jest komputerem, który w Polsce łatwo kupić na bazarze, w komisie, „z ogłoszenia". Instytucje mogą kupować go za pośrednictwem BOMIS-u, firm polonijnych, Domu Handlowego Nauki i innych.

ZX Spectrum przestał już być rewelacją (mimo, że pojawiają się zmodernizowane wersje — Spectrum +, a w 1986 r. ukazało się Spectrum ze 128 kbajtami pamięci). Jednak zainteresowania informatyką, mikrokomputerami (także najpopularniejszym z nich ZX Spectrum) jest nadal bardzo duże.

Wielu ludzi, którzy pierwotnie używali Spectrum do zabawy, zaczyna pisać własne programy — najpierw w Basicu, potem nawet w języku asemblera. Popularność zdobywa język Logo. Informatyka staje się nowym hobby.

Brak jest w Polsce książek, które przystępnie opisywałyby budowę i działanie komputerów osobistych, język Basic, itd. Różne czasopisma publikują co najwyżej artykuły na ten temat. Dla przykładu, jedyne znane autorom książki traktujące o ZX Spectrum, to bardzo drogie (lecz nie zawsze dobre i nie zawsze dostępne) tłumaczenia podręcznika użytkownika ZX Spectrum. Niektórzy korzystają z książek (lub kserokopii książek) wydanych w języku angielskim (rzadziej niemieckim lub francuskim), lecz obcojęzyczne książki nie są dostępne powszechnie.

Popularność ZX Spectrum w Polsce, duże zainteresowanie tematem, brak polskich książek o komputerach osobistych, wiedza nabyta w czasie użytkowania tego komputera oraz lektury kilkunastu zagranicznych pozycji, to wszystko skłoniło nas do napisania tej książki.

Książka przeznaczona jest dla osób zainteresowanych komputerami osobistymi, hobbistów, właścicieli i użytkowników ZX Spectrum. Mamy nadzieję, że będzie przydatna zarówno dla osób, które po raz pierwszy mają do czynienia z komputerem, jak i dla tych, którzy posiadają pewną wiedzę w tej dziedzinie. Stanowi ona kompromis między podręcznikiem opisującym komputer „od podstaw", a książką dla bardziej zaawansowanych Czytelników. Sądzimy, że zainteresuje ona szeroki krąg odbiorców.

Niektóre wiadomości powtórzone zostały w wielu miejscach książki, aby Czytelnik mógł je sobie lepiej utrwalić. Niejednokrotnie konieczne były odwołania do informacji zawartych w dalszych partiach, bowiem trudno rozstrzygnąć: co należy opisać najpierw. Do wyjaśnienia pewnych terminów potrzebna jest wiedza o innych. Stąd konieczność odwoływania się do fragmentów różnych rozdziałów.

Książka ta nie została pomyślana jako podręcznik programowania w języku Basic ZX Spectrum. Dlatego opis Basica zajmuje tylko dwa rozdziały (rozdz. 2 i 3). Wierzymy, że taka ilość informacji będzie jednak wystarczająca do posługiwania się tym językiem.

Przewodnik po ZX Spectrum jest vademecum wiedzy o tym komputerze. Chcielibyśmy, by był on jednocześnie instrukcją obsługi, podręcznikiem programowania, by uczył jak tworzyć grafikę, muzykę. By książka uczyła: jak posługiwać się komputerem, jak napisać własny program, by informowała, czym jest komputer.

Dewizą podróżników jest: ,,kto pyta nie błądzi". W książce tej staraliśmy się odpowiedzieć na pytania, które sami zadalibyśmy, gdyby przyszło nam poznawać ZX Spectrum raz jeszcze. Stąd nazwa przewodnik.

Próbowaliśmy także przekazać trochę ogólnej wiedzy o informatyce. Komputer bowiem stał się elementem kultury ludzkiej i wiedza o nim jest współczesnemu człowiekowi potrzebna.

W książce umieszczono wyjaśnienia wielu terminów podstawowych. Są one przeznaczone dla osób, które nie miały dotychczas styczności z komputerem. Brak miejsca nie pozwolił wyjaśnić dokładnie podstaw budowy, działania i programowania komputera, dlatego w wielu miejscach uciekano się do tłumaczeń intuicyjnych. Mamy nadzieję, że pomogą one w zrozumieniu książki. Jeśli nie będą wystarczające, Czytelnik powinien sięgnąć do literatury uzupełniającej.

Staraliśmy się także, by ,,Przewodnik..." mógł być pomocny dla osób, które dysponują pewną wiedzą informatyczną lub nawet stosują ZX Spectrum do pracy. Dla nich przeznaczone są szczegółowe informacje o sposobach współpracy z systemem operacyjnym, wykorzystaniu zmiennych systemowych i bardziej wyrafinowanych technikach programowania tego komputera.

Ceną, jaką przyszło zapłacić przyjmując koncepcję książki — vademecum dla każdego — jest jej niejednorodność. Często w tym samym rozdziale sąsiadują ze sobą wiadomości podstawowe i trudniejsze kwestie. Dlatego koniecznością stało się wskazanie rozdziałów łatwych (oznaczonych *), trudniejszych (**) i trudnych (***). Oznaczenia te pełnią rolę ,,Przewodnika po Przewodniku po ZX Spectrum". Mamy nadzieję, że osoby początkujące w miarę poznawania tajników informatyki będą mogły sięgać po trudniejsze partie tekstu.

Książkę tą podzielić można na trzy części:

CZĘŚĆ PIERWSZA zawiera podstawowe informacje o sposobie instalowania, użytkowania, budowie i programowaniu w języku Basic i składa się z rozdziałów: 1, 2, 3 i 5 (częściowo). Jest ona przeznaczona dla osób początkujących, choć są tu także fragmenty trudniejsze.

CZĘŚĆ DRUGA zawiera informacje o tym, jak uzyskać ,,więcej" z ZX Spectrum (rozdziały 4, częściowo 5 i 6). Jej lektura wymaga pewnej wiedzy mimo, że rozdz. 4 zawiera podstawowe pojęcia z dziedziny programowania w języku asemblera. Jest ona przeznaczona dla bardziej zaawansowanych Czytelników.

CZĘŚĆ TRZECIA to przegląd programów dostępnych na tym komputerze (języków programowania i programów użytkowych) oraz pamięci dyskowych (rozdz. 7, 8, 9). Jest ona adresowana do wszystkich Czytelników, chociaż rozdz. 8 jest trudniejszy.

Na zakończenie chcemy przeprosić za błędy, które zapewne znalazły się w książce (znane porzekadło informatyczne mówi, że... ,,w programie zawsze jest jeszcze jeden błąd").

* ZX Spectrum z zewnątrz i od środka 1

* Jak rozpocząć pracę z ZX Spectrum? 1.1

W zestawie ZX Spectrum znajduje się: mikrokomputer*), zasilacz, koncentryczny kabel antenowy, kabel łączący z magnetofonem, podręcznik do nauki programowania w języku Basic (wersja obcojęzyczna) oraz kaseta demonstracyjna.

Rozpoczynając pracę należy:

— włączyć telewizor oraz zasilacz mikrokomputera do sieci,
— włączyć telewizor,
— podłączyć kablem antenowym komputer (gniazdo TV) z wejściem antenowym telewizora,
— włączyć zasilanie komputera (gniazdo 9 V DC).

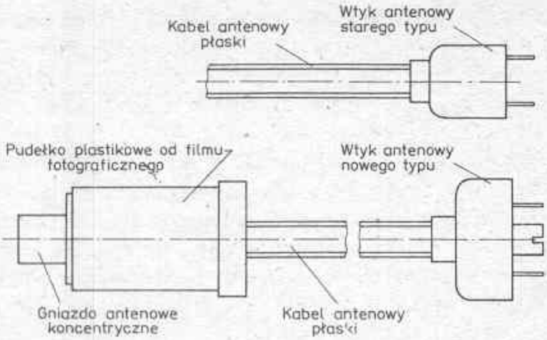
Ze względu na różne kształty gniazd trudno jest je pomylić.

Jeżeli pracę z komputerem rozpoczynamy po raz pierwszy, to po wykonaniu powyższych czynności należy dostroić telewizor. W tym celu wybieramy jeden z kanałów i przełączamy przełącznik zakresów (kanały 21÷60), następnie obracając odpowiednim pokrętłem wybieramy 36 kanał, czyli kanał, do współpracy z którym jest przystosowany ZX Spectrum. Telewizor należy dostrajać do momentu aż na ekranie pojawi się czytelny napis: „© 1982 Sinclair Research Ltd." Mikrokomputer jest gotowy do pracy.

Przy każdym następnym rozpoczęciu pracy z komputerem, po załączeniu zasilania, na ekranie telewizora pojawi się czarny prostokąt (nieraz z kilkoma kolorowymi kwadracikami). Po krótkiej chwili, po zniknięciu prostokąta, na dole ekranu ukaże się napis jak wyżej. ZX Spectrum czeka wtedy na wydanie mu polecenia.

Z pewnością część polskich użytkowników będzie wykorzystywała do współ-

*) Mikrokomputerem nazywa się komputer, który zawiera mikroprocesor (patrz 1.3). W książce tej oba te terminy będą stosowane dla określenia ZX Spectrum.



Rys. 1. Złącze przejściowe symetryczne — koncentryczne

pracy z ZX Spectrum telewizory czarno-białe. Pewna liczba tych telewizorów — starego typu — nie ma koncentrycznego wejścia antenowego, tzn. takiego gniazda, do którego pasuje wtyk antenowy komputerowego kabla. Należy wówczas wykonać własne złącze przejściowe — choćby takie jak na rys. 1. Bardzo stare modele telewizorów z reguły nie mogą odbierać sygnałów z tak wysokich kanałów. Konieczną jest wówczas głowica VHF (dostępna w handlu) lub niestety nowy telewizor.

Gniazda EAR oraz MIC na tylnej płycie mikrokomputera służą przede wszystkim do podłączenia magnetofonu (patrz rys. 6). Kabel połączeniowy ma dwie różnokolorowe pary końcówek, np. czarną i szarą. Wskazane jest przyporządkować końcówki jednego koloru do wejścia EAR oraz wyjścia słuchowego magnetofonu, a drugiego koloru do wyjścia MIC komputera i wejścia mikrofonowego magnetofonu. Pozwoli to na uniknięcie pomyłek w czasie wczytywania i zapisywania programów.

Chcąc wczytać program z taśmy magnetofonowej (np. demonstracyjnej) do pamięci komputera należy włączyć jedną z końcówek kabla połączeniowego do gniazda EAR, a drugą, tego samego koloru, do wyjścia słuchawkowego magnetofonu (z załączonym zasilaniem). Następnie potencjometry siły głosu i barwy tonu ustawiamy na środku skali i rozpoczynamy eksperymentalne dobranie poziomu sygnału. W tym celu należy:

— przewinąć taśmę do początku programu,
— ustawić poziom sygnału za pomocą potencjometru siły głosu,
— wprowadzić instrukcję LOAD (klawisz J), np. LOAD"", ENTER
— włączyć magnetofon.

Jeżeli siła głosu i barwa tonu są dobrze dobrane, to na ekranie telewizora — poza wydzielonym prostokątem — pojawią się na czas ok. 5 s niebiesko-czerwone paski. Po nich na ekranie ukaże się napis: „Program: *nazwa*", następnie ponownie niebiesko-czerwone paski, przez okres ok. 2 s, a zaraz po nich paski niebiesko-żółte, o większym zagęszczeniu, które świadczą o tym, że program ładuje się do pamięci komputera. W przypadku, gdy na ekranie nie ukaże się napis „Program:

nazwa" lub na dole ekranu pojawi się komunikat: ,,Tape loading error 0:1", należy zmienić poziom lub barwę tonu sygnału oraz powtórzyć wyżej wymienione czynności od początku. Instrukcja do ZX Spectrum zaleca, aby poziom sygnału był ustawiony na granicy słyszalności głośnika mikrokomputera. W przypadku stosowania takich magnetofonów jak: MK 232, M 101, jest to niemożliwe. Siłę głosu należy ustawić na minimum 2/3 skali.

Niejednokrotnie przyczyną kłopotów z wczytaniem programów do pamięci komputera jest to, że źle ekranowany (wewnątrz) telewizor oddziałuje na blisko stojący magnetofon i uniemożliwia wczytanie programu. Można temu przeciwdziałać przez ustawienie magnetofonu jak najdalej od telewizora. Jeśli to nie pomaga, to należy wyłączyć telewizor na czas wczytywania programu. Pozwoli to na zweryfikowanie przyczyny kłopotu.

Źle pracujący magnetofon znacznie utrudnia wczytywanie programów do pamięci komputera. Niekiedy ma on źle ustawioną głowicę lub rozregulowany układ przesuwu taśmy, powodujący nierównomierność jej przesuwu. O tym, czy magnetofon nadaje się do współpracy z komputerem użytkownik przekonuje się, próbując wczytać któryś z programów z dobrze nagranej kasety demonstracyjnej. Przy źle ustawionej głowicy mikrokomputer nie będzie reagował na sygnał z magnetofonu lub też będą znikały i pojawiały się ponownie czerwono-niebieskie paski, co doprowadzi do nie wczytania programu. Przy nierównomiernościach przesuwu taśmy niebiesko-czerwone paski, które powinny ,,stać" na ekranie lub powoli się przesuwać, będą poruszały się nierównomiernie. Nierównomierności przesuwu taśmy utrudniają lub uniemożliwiają wczytanie się programu. W takim przypadku należy naprawić lub zmienić magnetofon.

Jeżeli program zapisany na innym magnetofonie nie ładuje się do pamięci komputera, to często jedynym sposobem wczytania go jest użycie do wczytywania magnetofonu, na którym był on zapisany.

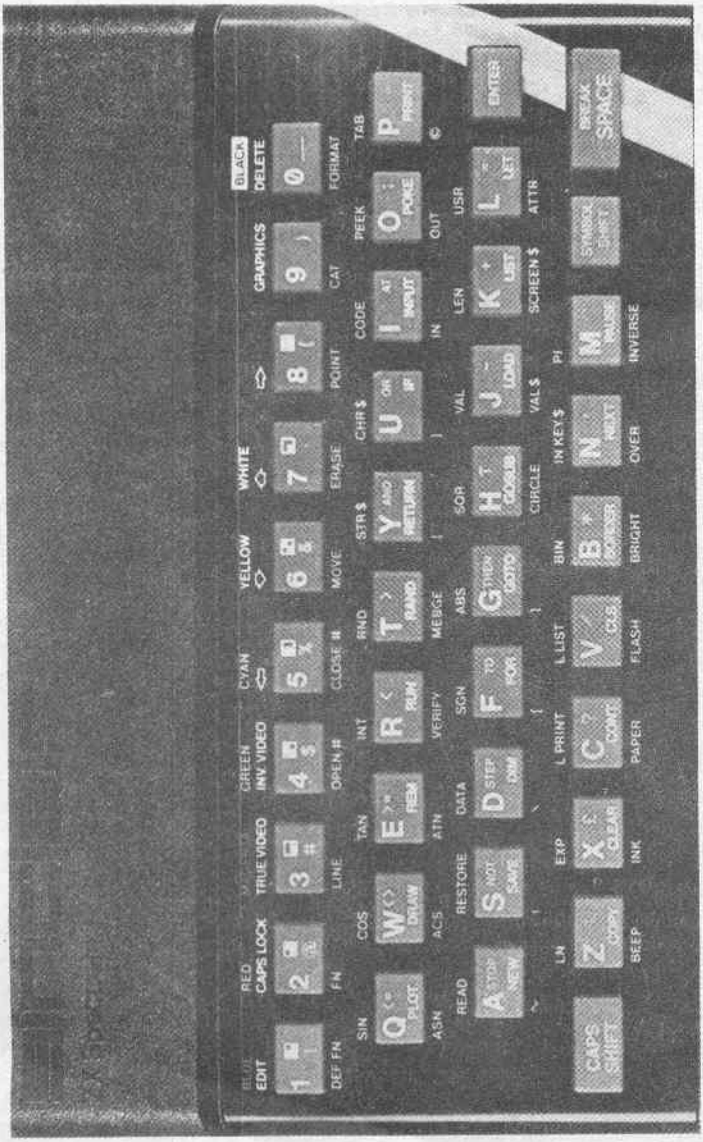
Wskazane jest by magnetofon współpracujący z komputerem miał licznik. Odszukanie programu na taśmie jest wtedy znacznie szybsze.

* Jak obsługiwać klawiaturę? 1.2

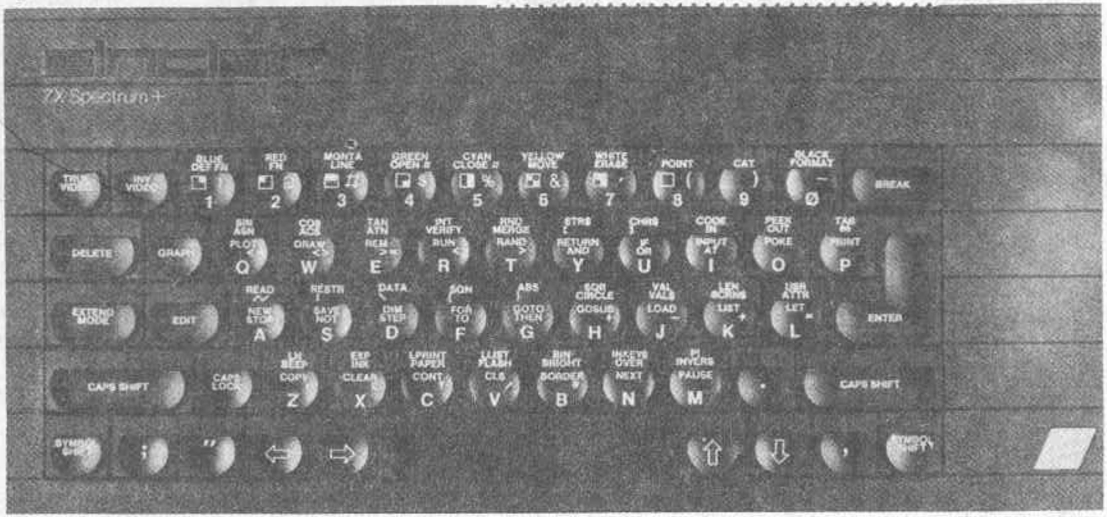
Posługiwanie się ZX Spectrum wymaga poznania klawiatury. Dlatego proponuje się szczególnie uważne przeczytanie tego punktu.

Pierwsze spojrzenie na klawiaturę ZX Spectrum może budzić przerażenie (rys. 2). Na szczęście jest to uczucie przejściowe. Bliższe przypatrzenie się klawiaturze pozwala dostrzec pewne prawidłowości w opisach poszczegolnych klawiszy. Poznanie ich znaczenia ułatwi pracę z mikrokomputerem.

ZX Spectrum ma 40 klawiszy. Na poszczególnych klawiszach są opisane nie tylko litery i cyfry, ale również słowa kluczowe języka Basic, nazwy funkcji, znaki interpunkcyjne oraz znaki dodatkowe takie jak nawiasy, procent, itd. Zdecydowana większość klawiszy ma po 5 znaczeń. Określone znaczenie danego klawisza wybiera się za pomocą klawiszy funkcyjnych SYMBOL SHIFT (ozna-



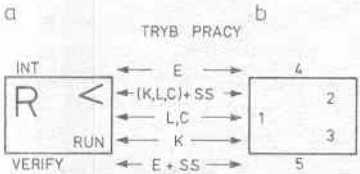
b



Rys. 2. Klawiatura:
a — ZX Spectrum, b — ZX Spectrum +

czanego dalej SS) lub CAPS SHIFT (CS). Poszczególne tryby pracy klawiatury omówiono posługując się przykładowym klawiszem. Każdemu ze znaczeń tego klawisza przyporządkowano cyfry od 1 do 5 (rys. 3).

Po włączeniu zasilania mikrokomputera, odczekaniu chwili i naciśnięciu klawisza ENTER lub SPACE w lewym dolnym rogu ekranu pojawia się migający kwadrat z literą K — tzw. k u r s o r — wskazujący, w którym miejscu zostanie wpisany znak (symbol z klawiatury). W trybie K (słowa kluczowe — ang. *Keyword*) są dostępne wszystkie polecenia języka Basic, opisane na danym klawiszu



Rys. 3. Klawisz pięciofunkcyjny:
a — przypisane instrukcje, b — oznaczenia umowne

w miejscu oznaczonym cyfrą 3. Dotyczy to trzech dolnych rzędów klawiatury. W przypadku klawisza z rys. 3a będzie to instrukcja RUN. W górnym rzędzie w trybie K dostępne są cyfry. Potrzebne są one m.in. do wprowadzania numerów linii programu w języku Basic.

Po wprowadzeniu słowa kluczowego tryb pracy klawiatury zmienia się automatycznie na L (od ang. *Letters* — litery). W tym trybie są dostępne wszystkie małe litery oznaczone na rys. 3b cyfrą 1, w tym przypadku *r*, oraz cyfry. Jeśli tekst ma być pisany dużymi literami należy nacisnąć CAPS SHIFT oraz dowolny klawisz z literą. Po zwolnieniu klawisza CAPS SHIFT ponownie będą dostępne, w trybie L, małe litery. Przełączenie na stałe z trybu L (małe litery) na C (duże litery) wykonuje się przez jednoczesne przyciśnięcie klawiszy CAPS SHIFT i klawisza z cyfrą 2. Na ekranie pojawi się migający kursor z literą C (C od ang. *Capital Letters*). Przy przełączeniu z trybu L na C oraz w każdym innym przypadku kiedy naciskane są jednocześnie dwa klawisze lub pojedynczy klawisz, należy pamiętać, że mają one określony czas repetycji. Każdy klawisz może być przyciśnięty przez ok. 0,7 s. Po upływie tego czasu następuje automatyczne powtórne wczytanie tego znaku lub słowa kluczowego, z częstotliwością 10 razy na sekundę (czas repetycji — 1/10 s). Zapomnienie o tym bywa przyczyną pomyłek*).

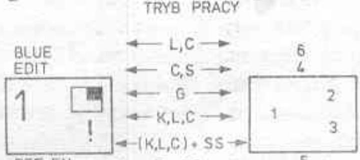
*) Czas, po którym następuje repetycja, można zmienić samemu, wpisując pod adres 23561 (jest to adres zmiennej systemowej nazwanej REPDEL) liczbę z przedziału 0—255. Normalnie, pod tym adresem wpisana jest liczba 35, co odpowiada 35/50 = 0,7 s. Czas repetycji, po upływie którego komputer uznaje, że klawisz został wciśnięty ponownie, określony jest wartością zmiennej systemowej zwanej REPPER, znajdującej się pod adresem 23562. Zazwyczaj pod tym adresem wpisana jest liczba 5. Wpisanie liczby większej od 5, lecz mniejszej od 256 wydłuża czas repetycji.

Wyjście z trybu C następuje po ponownym jednoczesnym wciśnięciu klawiszy CAPS SHIFT i 2. Jeżeli komputer pracuje w trybie K, L lub C, to po jednoczesnym wciśnięciu klawisza funkcyjnego SS oraz innego dowolnego klawisza otrzymamy na ekranie monitora znak narysowany na klawiszu kolorem czerwonym (niefortunnie w Spectrum + wszystkie oznaczenia na klawiszach wykonano kolorem białym), oznaczony na rys. 3b cyfrą 2. W naszym przykładzie będzie to znak <. Na klawiszach z cyframi wczytywany w tym trybie znak, znajduje się w miejscu oznaczonym cyfrą 3. Na klawiszu jest on również oznaczony kolorem czerwonym.

Kolejnym trybem pracy klawiatury ZX Spectrum jest tryb E (E od ang. *Extended mode*), czyli tryb rozszerzony. Pojawia się on po jednoczesnym naciśnięciu obydwu klawiszy funkcyjnych CAPS SHIFT i SYMBOL SHIFT. Dostępne są w nim symbole umieszczone nad klawiszami, oznaczone kolorem zielonym (pozycja 4 na rys. 3b — INT). Po wprowadzeniu wybranego symbolu tryb pracy klawiatury zmienia się z E na L (lub C).

Jeżeli w trybie E naciśniemy klawisz SYMBOL SHIFT, wówczas do komputera można wprowadzić symbole znajdujące się pod klawiszami, oznaczone kolorem czerwonym, pozycja 5 na rys. 3b. W przypadku klawisza z rys. 3a będzie to VERIFY.

Wszystko, co zostało do tej pory powiedziane, dotyczy trzech dolnych rzędów klawiatury, z wyjątkiem klawiszy ENTER oraz BREAK/SPACE. Górny rząd klawiatury ma trochę inne oznaczenia. O cyfrach i znakach w kolorze czerwonym, będących na i pod klawiszem, już powiedziano. Należy pamiętać jednak, że



Rys. 4. Klawisz sześciofunkcyjny:
a — przypisane instrukcje, b — oznaczenia umowne

polecenia napisane kolorem czerwonym pod klawiszami z cyframi 6, 7, 9, 0 mogą być wykonane tylko przy podłączeniu do ZX Spectrum układu Interface 1 (z microdriv'em).

Jako reprezentant górnego rzędu klawiatury zostanie omówiony klawisz z cyfrą 1 (rys. 4). Jednoczesne naciśnięcie CAPS SHIFT oraz 9 zmienia tryb pracy klawiatury na G (G od ang. *Graphics*) czyli na tryb graficzny. W trybie tym wprowadza się znaki graficzne narysowane na klawiszach z cyframi od 1 do 8. Znaki te mogą być także wyświetlane inwersyjnie, tzn., że kolor tła zmienia się z kolorem atramentu. Inwersyjny tryb wyświetlania znaków graficznych można najprościej otrzymać wciskając w trybie G klawisz CAPS SHIFT oraz jeden

z klawiszy od 1 do 8*). Wyjście z trybu G następuje po ponownym, jednoczesnym wciśnięciu CS oraz 9 lub tylko klawisza z cyfrą 9. Naciśnięcie CAPS SHIFT oraz jednego z klawiszy z cyframi powoduje wykonanie funkcji opisanej bezpośrednio nad nim białymi literami**).

Cyfry od 0 do 7 mogą definiować kolory, użyte jako argumenty instrukcji PAPER (kolor tła) oraz INK (kolor atramentu). Znaczenie ich jest opisane nad klawiszami odpowiednim kolorem, np. INK 1 powoduje wyświetlanie znaków na ekranie w kolorze ciemnoniebieskim***). Nad cyframi 8 i 9 nie ma opisów dotyczących kolorów, mają one specyficzne znaczenie. Argument 8 w instrukcji PAPER lub INK powoduje, że kolor papieru (tła) lub atramentu nie ulegają zmianie****). Natomiast argument 9 powoduje, że komputer sam dobiera do koloru tła kontrastowy kolor atramentu i odwrotnie.

W fazie poprawiania programu bardzo użyteczne są klawisze z cyframi 1, 5, 6, 7, 8 i 0. Gdy na ekranie telewizora są wyświetlane linie programu, to za pomocą CS i 6 lub CS i 7 (zgodnie z oznaczeniami) można poruszać się znacznikiem ▷ między poszczególnymi liniami programu. Znacznik zawsze znajduje się między numerem linii a pierwszą jej instrukcją. Użycie CS i 1 powoduje, że linia ze znacznikiem jest kopiowana do dolnej części ekranu i tam może być poprawiona lub zmieniona (patrz rozdz. 2.3 — edytor ekranowy).

W żądane miejsce wewnątrz poprawianej linii programu można się przesunąć po jednoczesnym wciśnięciu CS i 5 lub CS i 8 zgodnie z oznaczeniami kierunku nad klawiszami. Dany znak wewnątrz linii można kasować, gdy przesunie się kursor za wybrany znak, a następnie naciśnie się jednocześnie CS i 0, tzw. DELETE. Poprawioną linię umieszcza się ponownie w pamięci programu przez naciśnięcie ENTER.

W nowej wersji ZX Spectrum (ZX Spectrum +) klawiatura jest wzbogacona o dodatkowe klawisze. Zastępują one między innymi wszystkie funkcje, które są opisane białymi literami nad klawiszami z cyframi np.: EDIT, CAPS LOCK, TRUE VIDEO, INV VIDEO, GRAPHICS, DELETE, EXTENDED MODE, znaki sterujące kursorem oraz niektóre znaki interpunkcyjne: kropka, cudzysłów, średnik i przecinek.

Bezpośrednio z klawiatury można definiować kolor tła oraz kolor atramentu w czasie wprowadzania programu w Basicu do pamięci — bez wykorzystywania poleceń PAPER i INK. Zadeklarowanie aktualnych kolorów może nastąpić dopiero po wprowadzeniu numeru linii. W tym celu należy włączyć tryb E, naciskając jednocześnie CAPS SHIFT i SYMBOL SHIFT, a następnie przyciskając klawisze, od 0 do 7, definiuje się aktualny kolor tła w danej linii. Gdy jedno-

*) Znaki graficzne w trybie inwersyjnym można także otrzymać włączając INV VIDEO (CAPS SHIFT i 4) oraz naciskając dowolny klawisz z cyfrą od 1 do 8, w trybie graficznym.

**) W ZX Spectrum są osobne klawisze realizujące jednoczesne wciśnięcie CS oraz dowolnej cyfry.

***) W przypadku telewizora czarno-białego zamiast kolorów uzyskuje się różne odcienie szarości.

****) Dokładniejsze wyjaśnienie patrz p. 3.2.

Tablica 1. Znaczenie klawiszy górnego rzędu klawiatury

Tryb pracy		Oznaczenie danego klawisza									
E	SYMBOL SHIFT	DEF FN	FN	LINE	OPEN #	CLOSE #	MOVE	ERASE	POINT	CAT	FORMAT
	CAPS SHIFT	INK BLUE	INK BLACK	INK MAGENTA	INK GREEN	INK CYAN	INK YELLOW	INK WHITE	FLASH OFF	FLASH ON	INK BLACK
	—	PAPER BLUE	PAPER RED	PAPER MAGENTA	PAPER GREEN	PAPER CYAN	PAPER YELLOW	PAPER WHITE	NORMAL BRIGHT	EXTRA BRIGHT	PAPER BLACK
G	CAPS SHIFT	▙	▟	▄	▛	▌	▞	▖	█	GRAPHICS OFF	DELETE
	—	▝	▘	▀	▗	▐	▚	▜	□	GRAPHICS OFF	DELETE
K	CAPS SHIFT	EDIT	CAPS LOCK	TRUE VIDEO	INVERSE VIDEO	⇦	⇩	⇧	⇨	GRAPHICS ON	DELETE
L lub C	SYMBOL SHIFT	!	@	#	$	%	&	'	(	)	—
	—	1	2	3	4	5	6	7	8	9	0

cześnie z cyfrą wciśnięty jest klawisz funkcyjny CAPS SHIFT, to definiuje się aktualny kolor atramentu.

Klawisze z cyframi 9 i 8 w trybie E służą do włączania i wyłączania migotania (FLASH) — wówczas kolory tła i atramentu zamieniają się co ułamek sekundy. Przy jednoczesnym wciśnięciu CS oraz 9 lub 8 włącza się lub wyłącza podwyższona jasność obrazu — BRIGHT. Wszystkie funkcje górnego rzędu klawiatury opisano w tablicy 1.

Podsumowując wszystko co do tej pory zostało powiedziane, o trybach pracy klawiatury należy zapamiętać:

- w trybie K wprowadza się na ekran słowa kluczowe języka Basic (umieszczone na klawiszu, opisane kolorem białym) oraz cyfry,
- w trybie L wprowadza się wszystkie małe litery oraz cyfry,
- w trybie C wprowadza się wszystkie duże litery oraz cyfry,
- w trybach K, L i C z naciśniętym klawiszem SYMBOL SHIFT wprowadza się wszystkie znaki opisane na klawiszach kolorem czerwonym,
- w trybie E wprowadza się polecenia i funkcje standardowe języka Basic, opisane nad klawiszami kolorem zielonym,
- w trybie E, z wciśniętym klawiszem funkcyjnym SS, wprowadza się znaki i instrukcje, opisane pod klawiszami kolorem czerwonym,
- w trybie G wprowadza się znaki graficzne umieszczone na klawiszach z cyframi $1 \div 8$,
- przyciśnięcie CS oraz jednego z klawiszy $0 \div 9$ pozwala na korzystanie z poleceń opisanych nad górnym rzędem klawiatury kolorem białym.

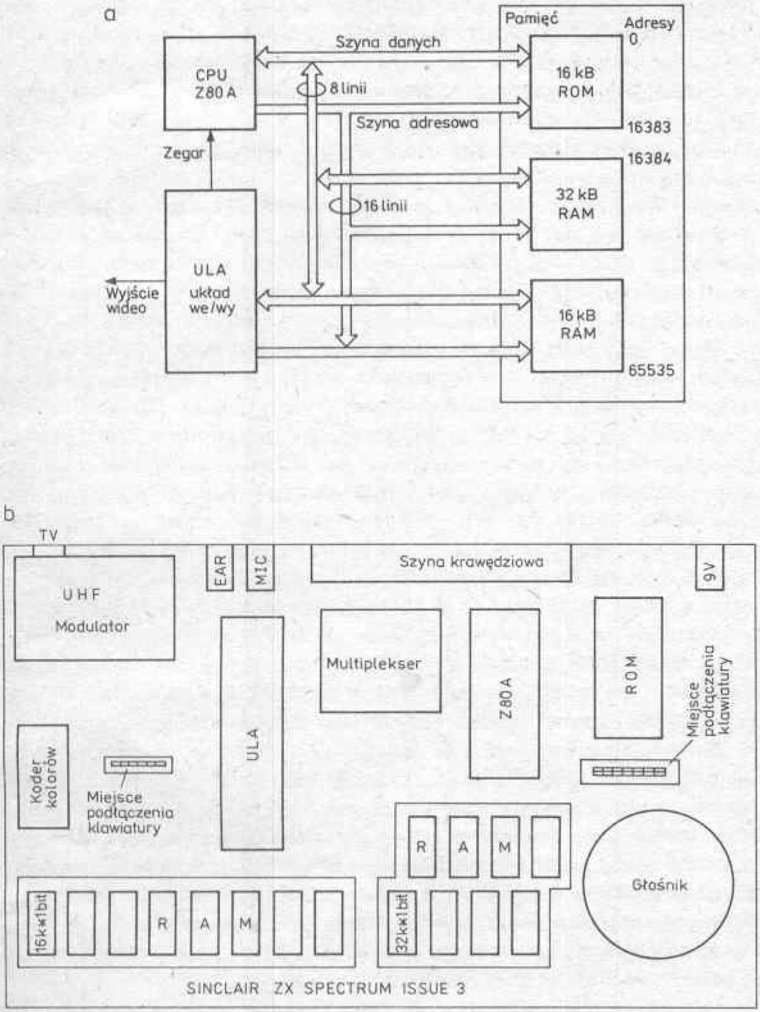
* * Wnętrze ZX Spectrum 1.3

Poznanie choć w stopniu podstawowym budowy mikrokomputera jest potrzebne po to, by lepiej zrozumieć jak on pracuje, choć nie jest to do jego użytkowania konieczne. Podane niżej informacje przeznaczone są dla osób posiadających podstawowe informacje z zakresu informatyki.

W zasadzie każdy komputer składa się z trzech podstawowych elementów: procesora (ang. *Central Processor Unit* — CPU), pamięci operacyjnej oraz układów pośredniczących w wymianie informacji między procesorem a otoczeniem, potocznie zwanymi układami wejścia-wyjścia (rys. 5).

,,Mózgiem" ZX Spectrum jest mikroprocesor Z80A firmy ZILOG, szybsza wersja znanego procesora Z80. Jedyną różnicą między tymi układami jest częstotliwość zegara. Dla Z80 wynosi ona 2,5 MHz, natomiast dla Z80A — 4 MHz (a w ZX Spectrum 3,5 MHz). Dzięki temu Z80A pracuje szybciej niż jego poprzednik.

Mikroprocesor zastosowany w ZX Spectrum ma ośmiobitową szynę danych, czyli 8 linii służących do wymiany informacji między CPU, pamięcią operacyjną i urządzeniami wejścia-wyjścia. Wymiana informacji odbywa się w postaci paczek po osiem bitów każda, czyli porcji składających się z kombinacji ośmiu zer i jedynek. Dlatego też mikroprocesor Z80 nazwano ośmiobitowym (jest także wiele



Rys. 5. ZX Spectrum:
a — schemat blokowy, b — ułożenie elementów na płytce

innych procesorów ośmiobitowych np.: 8080 firmy INTEL, 6502 firmy MOS TECHNOLOGY, czy 6800 firmy MOTOROLA).

Z80A ma szesnastobitową szynę adresową. Pozwala ona na wygenerowanie $2^{16}-1 = 65535$ różnych adresów. Adres wygenerowany przez mikroprocesor jednoznacznie wskazuje, która z komórek pamięci operacyjnej lub który z układów wejścia-wyjścia ma być w danej chwili zaangażowany w wymianę informacji. W praktyce, do wybierania układów wejścia-wyjścia wykorzystuje się osiem bitów, gdyż rzadko zdarza się, aby system wykorzystywał więcej niż 256 układów zewnętrznych wejścia-wyjścia.

Inną ważną funkcją procesora jest wytwarzanie sygnałów wskazujących, czy wygenerowany adres został przeznaczony dla pamięci czy też dla układów wejścia-wyjścia. Jeżeli wraz z adresem na odpowiednim wyjściu mikroprocesora pojawi się sygnał MREQ (*Memory REQuest*), to znaczy, że w przesłaniu informacji weźmie udział pamięć operacyjna. Natomiast, jeżeli z adresem pojawi się sygnał IORQ (*Input Output ReQuest*), to znaczy, że procesor żąda dostępu do jednego z układów wejścia-wyjścia. Należy pamiętać, że szyna adresowa przesyła sygnały tylko z mikroprocesora do układów wejścia-wyjścia lub do pamięci — nigdy odwrotnie. Inaczej rzecz się ma z szyną danych. Jest ona dwukierunkowa. Można nią przesyłać informacje z i do mikroprocesora. O tym, w którym kierunku ma nastąpić przesłanie decydują sygnały: RD (*ReaD*) — odczytu z pamięci lub układów wejścia-wyjścia, lub WR (*WRite*) — zapisu do pamięci lub przesłania na wyjście. Dokładniejsze informacje znajdują się w pracach [4, 11, 20, 21].

Pamięć Spectrum można podzielić na dwie podstawowe części: pamięć tylko do odczytu ROM (ang. *Read Only Memory*) i pamięć o dostępie swobodnym RAM (ang. *Random Access Memory*). Pamięć ROM zajmuje 16 kB (kilobajtów*)) od adresu 0 do 16383. Znajduje się tam specjalny program tzw. system operacyjny. Zawiera on m.in. procedury obsługi klawiatury, głośnika, współpracy z magnetofonem, obsługi ekranu, edytor, itd. Oprócz systemu operacyjnego ROM zawiera interpreter języka Basic. Jego zadaniem jest wykonywanie programów napisanych w języku Basic. Programy znajdujące się w pamięci ROM są napisane w kodzie maszynowym Z80. Ostatnie 768 bajtów ROM-u, od adresu 15616 do 16383, są przeznaczone na generator znaków (ang. *Character Generator*). Są tu zdefiniowane kształty wszystkich liter, cyfr i innych znaków, dostępnych na klawiaturze (patrz dodatek A; kody ASCII). Wszystkim znakom przyporządkowano liczby, tzw. kody, zgodne z przyjętym standardem ASCII. Bezpośrednio z klawiatury nie są dostępne polskie litery takie jak np. ą, ć, ń, ę (chociaż takie znaki można zdefiniować samemu).

ROM jest pamięcią, która służy do odczytu. Jest to tzw. pamięć stała, nie możemy nic w niej zmienić. Cechą charakterystyczną systemu operacyjnego umieszczonego w pamięci stałej ZX Spectrum jest to, że po włączeniu zasilania następuje automatyczny start programu przygotowującego komputer do pracy i już po chwili mikrokomputer jest gotowy do przyjmowania i wykonywania poleceń.

*) 1 kilobajt = 1 kB = 1024 bajty — jednostka ilości pamięci.

W wersji podstawowej pamięć RAM zajmuje 16 kB. Jednakże produkcja takiej wersji została już przez firmę Sinclair zaniechana i pamięć RAM ZX Spectrum jest rozszerzona do 48 kB (stąd też w nazwie mamy ZX Spectrum 48 k).

Właściciel ZX Spectrum 16 kB może samodzielnie rozszerzyć pamięć do 48 kB, gdyż na płytce wewnątrz mikrokomputera znajduje się zarezerwowane miejsce na dodatkowe układy scalone pamięci. Ostatnio wprowadzono na rynek nową wersję komputera z pamięcią 128 kB.

Najważniejszą cechą pamięci RAM jest możliwość zmiany zawartości jej komórek. Każdy program napisany przez użytkownika lub wczytany z pamięci zewnętrznej (np. z magnetofonu) oraz dane są umieszczane w tej części pamięci. Jeżeli odłączy się od mikrokomputera zasilanie lub stanie się to z przyczyn od nas niezależnych, wtedy cała zawartość pamięci RAM ulegnie bezpowrotnemu zniszczeniu, przeciwnie niż pamięci ROM.

Mimo, że cała pamięć RAM może być zmieniana, to istnieją pewne obszary (zmienne systemowe, mapy microdrive, tzw. UDG (patrz 3-5), itp.), w których po załączeniu zasilania system operacyjny umieszcza pewne dane, potrzebne mu do pracy. Niektóre z tych wartości możemy zmienić sami, co niekiedy bywa bardzo użyteczne. Należy zrobić to rozważnie, po wcześniejszym zapoznaniu się z systemem operacyjnym ZX Spectrum.

Bardzo ważnym elementem ZX Spectrum jest układ scalony o nazwie ULA (ang. *Uncommited Logic Array*). Można go porównać do dużego węzła łączności. Steruje on współpracą mikroprocesora z urządzeniami zewnętrznymi w taki sposób, aby nie nastąpiła „kolizja". ULA jest połączona z mikroprocesorem jako układ wejścia-wyjścia o adresie 254 (bit zerowy pierwszego bajtu adresu jest równy zero). Jest ona także połączona z szyną danych ZX Spectrum. Na przykład do jednego z wyprowadzeń ULA jest podłączony głośnik. ULA zajmuje się również kontrolą koloru brzegu ekranu. Do tej samej linii szyny danych co głośnik podłączony jest układ przeznaczony do współpracy z magnetofonem.

Podsumowując: szyna danych podłączona jest do ULA w ten sposób, że bity 0, 1, 2 sterują kolorem granicznej części ekranu. Bity 3 i 4 kontrolują wyjście z komputera MIC (wyjście na magnetofon) oraz głośnik.

ULA może pracować także jako urządzenie wejściowe. Steruje ona przesyłaniem danych z gniazdka EAR (wejście z magnetofonu) do pamięci mikrokomputera oraz jest urządzeniem pośredniczącym między mikroprocesorem, pamięcią a klawiaturą.

ULA odczytuje także informacje z pamięci obrazu (pewnego obszaru RAM) i na ich podstawie generuje kolorowy obraz telewizyjny, sterując pracą kodera PAL i modulatora VHF. Odbywa się to z częstotliwością 50 Hz. Dlatego mamy wrażenie, iż obraz jest stały, nie migający.

Wszystkie omówione wyżej — prócz ULA — elementy mikrokomputera są dostępne na rynkach krajów socjalistycznych. ULA natomiast jest układem wykonanym na zamówienie Sinclaira. Tego typu układy jak ULA bywają niekiedy bardziej skomplikowane niż mikroprocesory. Zdecydowana większość komputerów domowych ma choć jeden wyspecjalizowany układ podobny do ULA. Jest to więc jeden z najbardziej istotnych elementów mikrokomputera.

Mikrokomputer bez urządzeń zewnętrznych jest mało przydatny. Urządzenia peryferyjne takie jak: klawiatura, telewizor (monitor), drukarka, pamięć masowa itd. stwarzają możliwość komunikacji między użytkownikiem a mikrokomputerem. Konwersację z komputerem umożliwia klawiatura, będąca integralną częścią ZX Spectrum. Za jej pomocą pisane są programy, wydawane są komputerowi polecenia i wczytywane są dane potrzebne do pracy konkretnego programu.

Komputer komunikuje się z użytkownikiem za pośrednictwem telewizora. Informacje przeznaczone dla użytkownika są wyświetlane na ekranie. Telewizor musi odbierać 36 kanał (zrobiono tak dla wygody użytkowników, ponieważ w Wielkiej Brytanii program BBC II jest emitowany na tym kanale, a w tym właśnie kraju jest produkowany ZX Spectrum). Obecnie wszystkie produkowane w Polsce telewizory pracują w zakresie częstotliwości obejmującym 36 kanał.

ZX Spectrum generuje kolorowy obraz. Kodowanie kolorów odbywa się w systemie PAL*). W Polsce audycje telewizyjne są nadawane w systemie SECAM. Wynika z tego, że do otrzymania kolorowego obrazu należy posiadać telewizor z dekoderem PAL. Innym wyjściem z sytuacji jest przystosowanie naszego kolorowego telewizora do odbioru sygnałów w systemie PAL za pomocą odpowiedniego układu elektronicznego — dekodera PAL — dołączonego wewnątrz telewizora. Produkowane w naszym kraju (w kooperacji) telewizory, takie jak: Videoton, Colorette, Helios**) i Venus**) mają taki dekoder (Jowisz i Rubin nie — na ekranach tych telewizorów, podobnie jak na telewizorach czarno-białych, otrzymuje się obraz monochromatyczny).

Każdy mikrokomputer musi mieć możliwość współpracy z pamięcią zewnętrzną. Pamięcią tą może być zwykły magnetofon kasetowy. Programy zapisuje się na normalnej taśmie w ten sposób, w jaki nagrywa się muzykę. Kasety, na które są nagrywane programy, powinny być dobrej jakości i nie mogą mieć sklejeń. W przeciwnym razie powstaną przekłamania przy zapisie (lub odtwarzaniu) programu.

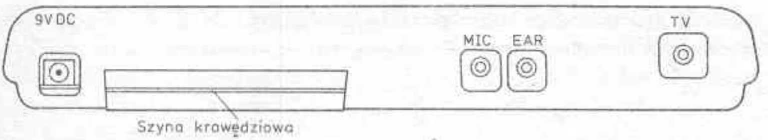
Średnia prędkość zapisu informacji (inna jest prędkość transmisji zer a inna jedynek) z pamięci komputera na taśmę, lub odwrotnie, wynosi około 1500 bodów (bit/s). Pamięć RAM ma 48 kB (49152 bajty), biorąc to pod uwagę transmisja jednego bloku programu o tej długości trwa około 320 s.

Jak wcześniej zostało powiedziane, wewnątrz ZX Spectrum znajduje się mały głośnik, spełniający jedynie funkcję pomocniczą. Sygnał podawany do głośnika będzie lepiej słychać, gdy wyjście MIC mikrokomputera połączy się z wejściem magnetofonowym wzmacniacza. Możliwe jest także podanie tego sygnału na wzmacniacz fonii telewizora.

Telewizor, magnetofon i wzmacniacz oraz głośnik zewnętrzny są podłączone bezpośrednio do odpowiednich wejść-wyjść ZX Spectrum (TV, EAR i MIC) —

*) We Francji można kupić ZX Spectrum pracujące w innej niż w Polsce wersji systemu SECAM.

**) Nie wszystkie typy.



Rys. 6. Tylna ścianka ZX Spectrum

rys. 6. Nie są to jedyne urządzenia mogące współpracować z tym komputerem domowym. Wszystkie sygnały używane przez procesor i urządzenia zewnętrzne są wyprowadzone na zewnątrz komputera w postaci łącza krawędziowego (ang. *Edge Connector* — patrz dodatek F). Do szyny tej można podłączyć inne niż wymienione urządzenia peryferyjne, bez konieczności dostawania się do wnętrza mikrokomputera. Sprzęga się je z systemem za pomocą odpowiednich urządzeń dopasowywujących zwanych interfejsami (z ang. *interface*). Każdy interfejs składa się ze sprzętu (ang. *hardware*) i oprogramowania (ang. *software*). Na ogół im bardziej jest skomplikowana część sprzętowa — tym mniej skomplikowana część programowa, i odwrotnie. W ten sposób do ZX Spectrum można dołączyć np. dowolną drukarę (z wyjątkiem ZX Printer, Seikosha GP 50 A, itp., które nie wymagają stosowania dodatkowego interfejsu).

Firma Sinclair jako podstawowe wyposażenie dodatkowe do swoich mikrokomputerów sprzedaje tzw. microdrive, który podłącza się do Spectrum za pomocą układu o nazwie Interface 1. Microdrive jest pamięcią masową, można go porównać do stacji dysków elastycznych. Informacje zapamiętywane są na pętli taśmy magnetycznej umieszczonej w kasecie. Obsługa microdrive jest dokonywana za pomocą programów umieszczonych w dodatkowej pamięci ROM zawartej w układzie Interface 1. Układ ten w pewien sposób stwierdza [5, 11], kiedy należy „podmienić" standardowy ROM dodatkową pamięcią stałą. Odwołanie do dodatkowego ROM-u następuje m.in. przez standardową procedurę obsługi błędu umieszczoną w systemie operacyjnym mikrokomputera.

Microdrive pozwala na znacznie szybsze wczytanie do pamięci ZX Spectrum lub zapisanie na taśmie żądanej informacji. Program o długości 48 kB wpisuje się do pamięci mikrokomputera (lub na kasetkę microdrive) kilkadziesiąt sekund, a nie ponad pięć minut jak to ma miejsce przy korzystaniu z magnetofonu. Szczególnie widoczne korzyści daje to przy uruchamianiu długich programów pisanych w kodzie maszynowym. Zrobienie błędu w takim programie często powoduje zawieszenie się systemu i jedynym wyjściem jest „wyczyszczenie" pamięci RAM przez krótkotrwałe odłączenie zasilania lub skorzystanie z klawisza RESET — o ile komputer go ma. Poza tym microdrive pozwala na dostęp do dowolnej informacji zapisanej na kasetce.

Interface 1 służy nie tylko jako pakiet dopasowujący instalowany między mikrokomputerem a microdrive. Urządzenie to pozwala na komunikację z różnymi urządzeniami w standardzie RS 232 (drukarkami, terminalami czy innymi komputerami, które mogą być podłączone bezpośrednio do Interface 1). Do podłączenia manipulatora używa się np. układu Interface 2.

Innym zastosowaniem Interface 1 jest wykorzystanie go do tworzenia lokalnych sieci komputerowych. W lokalną sieć komputerową (patrz rozdz. 6) można połączyć od 2 do 64 ZX Spectrum. Sieć lokalna ułatwia szybką wymianę informacji między poszczególnymi mikrokomputerami. Ma także zastosowanie przy rozwiązywaniu problemów wielozadaniowych. Wówczas każdy komputer sieci lokalnej wykonuje określone zadanie, a wyniki są przesyłane do wytypowanej jednostki, zajmującej się obróbką finalną. W układzie tym istnieje także możliwość sterowania pracą całej sieci za pomocą jednego mikrokomputera.

Sieci komputerowe mogą być wykorzystywane do organizowania komputerowych laboratoriów w szkołach. W takiej pracowni nauczyciel mógłby żądać dostępu do pamięci dowolnego komputera z sieci, a przez to kontrolować pracę ucznia. Kolejną zaletą sieci jest wykorzystanie przez wszystkie pozostałe komputery sieci szybkiej pamięci zewnętrznej (np. dysków elastycznych) oraz drukarki podłączonych do jednego z komputerów.

Dużo zastrzeżeń wśród użytkowników ZX Spectrum budzi klawiatura. Jest ona niewygodna w obsłudze. Do ZX Spectrum można podłączyć dodatkową klawiaturę. Klawiatury takie oferuje wiele firm. Jeżeli ktoś zna się choć trochę na elektronice, może ten problem rozwiązać we własnym zakresie, korzystając z rad zawartych w pracy [4]. Dodatkowa klawiatura przydaje się szczególnie wtedy, gdy z jednego mikrokomputera korzysta wiele osób. Ma to miejsce w klubach mikrokomputerowych lub podczas zajęć z wykorzystaniem tego sprzętu w szkołach czy na uczelniach. Dodatkowa klawiatura przedłuża żywotność ZX Spectrum. Porównując układ klawiatury ZX Spectrum z układami klawiatur polskich maszyn do pisania można zauważyć, że różnicą między nimi jest zamiana miejscami klawiszy Y z Z. Ma ona szczególne znaczenie, gdy mikrokomputer wykorzystuje się jako inteligentną maszynę do pisania.

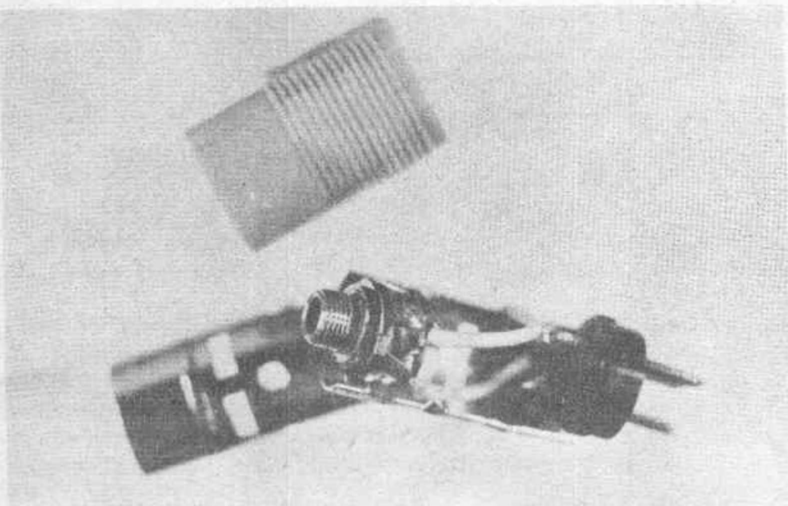
Omawiając pokrótce urządzenia zewnętrzne do ZX Spectrum, należy wspomnieć o piórze świetlnym (ang. *Light pen*). Pióro takie pozwala na rysowanie na ekranie telewizora barwnych obrazów. Można nim również ,,wycierać'' nieudane projekty graficzne, bez pozostawiania śladów. Zaprojektowany rysunek może zostać nagrany (zapamiętany) na taśmie magnetofonowej czy kasecie microdrive lub wydrukowany na drukarce. Dzięki wygodzie użycia pióra świetlne zyskały wielu zwolenników.

Przy grach oraz niektórych programach symulacyjnych często wykorzystuje się manipulatory (ang. *joystick*). Pozwalają one na sprawniejsze sterowanie mogącym poruszać się po ekranie obiektem. W większości przypadków realizują następujące funkcje: ruch w lewo, w prawo, do góry, na dół oraz zatrzymanie się lub strzał. Funkcje niektórych manipulatorów można definiować samemu.

Powyżej zostały omówione tylko wybrane urządzenia zewnętrzne ZX Spectrum. Wraz z upływem miesięcy wzrasta liczba urządzeń zewnętrznych do mikrokomputerów. Przekonać się o tym można studiując literaturę (np. czasopisma *Sinclair User* czy *Your Computer*).

* Jak zapisywać program w pamięci zewnętrznej? 1.5

Do przekazywania programów z pamięci ZX Spectrum na taśmę magnetofonową służy wyjście MIC, które łączymy z wejściem mikrofonowym magnetofonu. Końcówki kabla łączącego komputer z magnetofonem są wykonane w standardzie mini JACK. Natomiast wejścia mikrofonowe zdecydowanej większości magnetofonów produkowanych w Polsce są wykonane w standardzie DIN. Dlatego często niemożliwe jest bezpośrednie wykorzystanie do tego celu kabla połączeniowego, będącego w zestawie. Należy wówczas wykonać złącze pośrednie. Na rysunku 7 pokazano jedną z możliwych konstrukcji takiego złącza. Złącze to można umieścić w obudowie typowej wtyczki do gniazda mikrofonowego, co zapewni mu większą trwałość i estetyczny wygląd. Inną metodą jest zamontowanie dodatkowego gniazda bezpośrednio w obudowie magnetofonu, np. MK 232p. Wymaga to jednak pewnej znajomości elektroniki.



Rys. 7. Złącze pośrednie do współpracy z magnetofonem

Chcąc zapisać na taśmie wpisany do pamięci komputera program należy:
- znaleźć wolne miejsce na taśmie,
- ustalić odpowiedni poziom sygnału zapisu lub włączyć zapis automatyczny (patrz p. 1.1),
- połączyć gniazdo mikrokomputera MIC z gniazdem mikrofonowym magnetofonu,
- napisać, korzystając z klawiatury, polecenie SAVE "*nazwa*" lub SAVE "*nazwa*" LINE *nr linii*,

— nacisnąć klawisz ENTER,
— włączyć zapis w magnetofonie,
— nacisnąć dowolny klawisz.

Po zakończeniu nagrywania programu na taśmę w dolnej części ekranu pojawi się napis „OK 0:1". Jest to znak, że można wyłączyć magnetofon. Następnie nagrany program można zweryfikować, co daje pewność, że zapis jest prawidłowy. Weryfikacja odbywa się według następującego schematu:

— przewinąć taśmę na początek programu,
— połączyć gniazdo EAR mikrokomputera z wyjściem słuchawkowym magnetofonu,
— wpisać polecenie VERIFY" "lub VERIFY „*nazwa*",
— nacisnąć klawisz ENTER,
— włączyć magnetofon.

Jeżeli program został zapisany na taśmie prawidłowo, to na dole ekranu pojawi się napis „OK 0:1". Jeżeli program nie daje się zweryfikować, to należy zmienić poziomu sygnału i wykonać powyższe czynności ponownie. Jeżeli to nie da spodziewanego rezultatu, to należy zapisać jeszcze raz program na taśmie — niejednokrotnie w innym miejscu, gdyż wady ukryte taśmy magnetofonowej również mogą być przyczyną niewłaściwego zapisu. W celu zwiększenia pewności, że program o dużym znaczeniu dla użytkownika nie ulegnie przypadkowemu zniszczeniu, należy nagrać go kilkakrotnie. Często robi się to na dwu różnych taśmach, wówczas nawet po przypadkowym skasowaniu fragmentów programu, może on być odtworzony z innej kopii.

Omawiając współpracę ZX Spectrum z magnetofonem wspomniano instrukcje języka Basic przeznaczone do jego obsługi. Ponieważ użytkownik może korzystać z gotowych programów zanim będzie potrafił napisać własne, zdecydowano podać instrukcje obsługi magnetofonu przed opisem całego języka. Programów (np. gier) można używać wczytując je z magnetofonu za pomocą instrukcji:

```
LOAD     "nazwa programu"
  ↑       ↑
  |       klawisz P
  |
klawisz J
```

Zwykle programy takie automatycznie zaczynają działać.

Dla wygody korzystania z tej książki postanowiono w jednym miejscu zebrać wszystkie informacje o współpracy komputera z magnetofonem. ZX Spectrum ma cztery instrukcje do współpracy z magnetofonem:

LOAD, MERGE — wczytywanie z taśmy do pamięci komputera,

SAVE — zapisywanie na taśmę,

VERIFY — weryfikacja poprawności zapisu na taśmę.

Na taśmie zapisać można program w Basicu. Zapamiętuje się wtedy obszar programu i zmiennych (patrz rozdz. 2.5), tablicę używaną w języku Basic, obszar pamięci ekranu (patrz rozdz. 3.4) lub dowolny obszar pamięci, np. program binarny.

- Zapis programu w Basicu na taśmie magnetofonowej:

  SAVE "*nazwa*"

  ↑ nazwa, pod którą program ma być zapisany na taśmie

lub SAVE "*nazwa*" LINE *numer*

  ↑ numer linii, od której rozpocznie się automatycznie działanie programu po wczytaniu go do pamięci.

- Zweryfikowanie poprawności zapisu. Program zapisany na taśmie jest odczytywany i porównywany z zawartością pamięci (jeśli różnią się one sygnalizowany jest błąd; należy wtedy program zapisać na taśmie ponownie):

  VERIFY "*nazwa*"

  ↑ nazwa programu na taśmie, z którym ma być porównywana zawartość pamięci. Gdy zamiast nazwy poda się " "(nazwę pustą tzn. dwa cudzysłowy jeden za drugim) porównywany jest pierwszy program napotkany na taśmie.

- Wczytanie programu do pamięci:

  LOAD "*nazwa*"

  ↑ nazwa programu, który ma być wczytany lub nazwa pusta (" ") — analogicznie jak w VERIFY.

- Dołączenie programu. Instrukcja ta dopisuje do programu znajdującego się w pamięci program znajdujący się na taśmie. Jeśli w starym i nowym programie znajdują się linie o tych samych numerach, to stare linie są usuwane i zastępowane nowymi.

  MERGE "*nazwa*"

  ↑ nazwa programu lub (" "), analogicznie jak w VERIFY.

Tablice używane przez program w języku Basic

- Zapamiętanie tablicy na taśmie:

  SAVE "*nazwa*" DATA *nazwa tablicy* ( )

"*nazwa*" ↑	*nazwa tablicy* ↑	( ) ↑
nazwa jaką będzie miała tablica zapisana na taśmie	nazwa tablicy używanej przez program w języku Basic (jednoliterowa)	znaki nawiasów — konieczne

- Wczytanie do pamięci:

  LOAD "*nazwa*" DATA *nazwa tablicy* ( )

  ↑ nazwa może być pusta

- Zweryfikowanie zapisu:

  VERIFY "*nazwa*" DATA *nazwa tablicy* ( )

Program binarny

- Zapisanie na taśmie bloku pamięci:

  SAVE "*nazwa*" CODE *adres*, *długość*

  ↑ nazwa bloku na taśmie

  ↑ adres miejsca pamięci, od którego zaczyna się blok

  ↑ długość bloku (w bajtach)

- Zweryfikowanie zapisu:

  VERIFY "*nazwa*" CODE *adres*, *długość*

  ↑ adres i długość*) można pominąć tzn. napisać VERIFY "*nazwa*" CODE. W takim przypadku przyjęte zostaną wartości jakich użyto w instrukcji SAVE.

- Wczytanie z taśmy:

  LOAD "*nazwa*" CODE *adres*, *długość*

  ↑ nazwa może być pusta

  ↑ ↑ długość ładowanego bloku*) znaczenie parametrów jak w instrukcji SAVE CODE. Adres informuje, w którym miejscu pamięci należy umieścić blok.
  Jeśli parametry pominie się, to zostaną przyjęte wartości podane w instrukcji SAVE CODE.
  Gdy poda się jeden, to oznacza on adres.

- Instrukcja:

  SAVE "*nazwa*" SCREEN $

  powoduje zapisanie na taśmie obrazu widocznego na ekranie. Jest ona równoważna instrukcji.

  SAVE "*nazwa*" CODE 16384, 6912

- Instrukcja

  LOAD "*nazwa*" SCREEN $

  ↑ może być nazwa pusta

  powoduje „wczytanie obrazu" z taśmy. Zapamiętany obraz ukaże się na ekranie.

*) Jeżeli parametr długość jest mniejszy od podanego w instrukcji SAVE, to wystąpi błąd ładowania. Natomiast może on być większy.

Instrukcja ta jest równoważna:

LOAD "*nazwa*" CODE 16384, 6912

Magnetofon nie jest jedynym rodzajem pamięci masowej. Może on zostać z powodzeniem zastąpiony przez urządzenia typu microdrive lub stację dysków. W przypadku stosowania microdrive jako pamięci masowej opisane polecenia LOAD, SAVE, VERIFY, MERGE mają inną postać (patrz p. 6.2.1).

Z taśmami lub kasetkami microdrive trzeba postępować właściwie. Pozwala to uchronić zawarte na nich informacje przed przypadkowym zniszczeniem lub znacznie przedłużyć ich żywotność. Oto kilka praktycznych rad:

— rzadko używane taśmy z nagranymi programami należy co pewien czas (np. co 3 miesiące) przewinąć, aby zniszczyć powstałe wypadkowe pole magnetyczne,
— jeżeli wiele kaset przechowywanych jest razem, to powinno się oddzielać je od siebie warstwą cienkiej folii aluminiowej,
— taśmy i kasetki nie mogą być narażone na silne oddziaływanie promieni słonecznych i ciepła, gdyż może to doprowadzić do ich rozmagnesowania,
— taśm nie wolno dotykać palcami,
— nie wolno nigdy wyjmować kasetki z microdrive podczas odczytu lub zapisu, tzn. gdy na przedniej ścianie w microdrive pali się czerwone światełko,
— nie należy wyłączać zasilania komputera, podczas gdy w microdrive znajduje się kasetka,
— microdrive muszą być odpowiednio konserwowane i chronione przed uszkodzeniami mechanicznymi.

Stosując powyższe wskazówki w większości przypadków można uniknąć przykrych rozczarowań, związanych z bezpowrotną utratą zapisanych na taśmie informacji.

Niżej podano prosty sposób pozwalający na skatalogowanie programów zapisanych na taśmie. Należy w tym celu:

— połączyć komputer z telewizorem i magnetofonem (nie zapominając o zasilaniu),
— przewinąć taśmę do jej początku,
— wyzerować licznik magnetofonu,
— napisać korzystając z klawiatury np.: LOAD "X1754Y" (nazwa, jakiej nie ma żaden z nagranych na taśmie programów); zamiast LOAD można użyć MERGE lub VERIFY,
— nacisnąć klawisz ENTER,
— włączyć magnetofon.

W czasie wyświetlenia na ekranie tytułów wszystkich segmentów, z których składa się program, należy zanotować przy danej nazwie aktualny stan licznika magnetofonu. Na tak opisanej taśmie programy odszukuje się łatwo i szybko.

* Kilka rad praktycznych 1.6

Na rysunku 7 przedstawiono złącze dopasowujące przeznaczone do nagrywania programów na taśmę z pamięci komputera. Jest ono przystosowane do współ-

pracy z takimi magnetofonami, jak: MK 232p, M 101, czy innym, produkowanym w Polsce, magnetofonem. Przy wczytywaniu programów z magnetofonu do pamięci komputera, takie układy też mogą być potrzebne. Może to mieć miejsce w dwu zasadniczych przypadkach:

— posiadamy magnetofon z wyjściem słuchawkowym lub głośnikowym o standardzie innym niż JACK,
— współpracujemy np. z magnetofonem M 101.

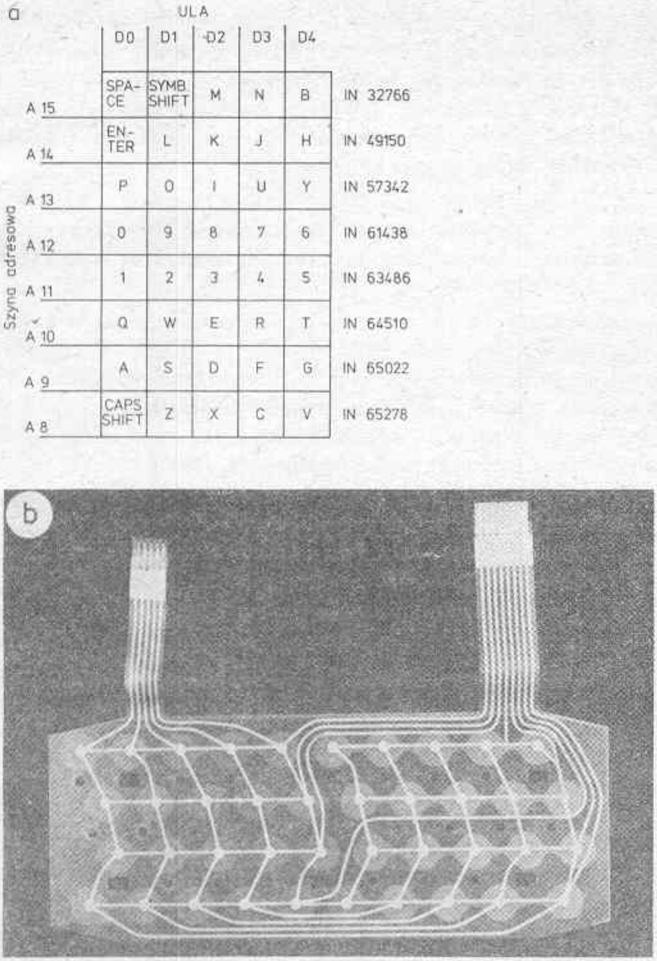
W pierwszym przypadku należy wykonać dodatkowy kabel zakończony z jednej strony wtykiem JACK, a z drugiej — wtykiem odpowiedniego typu lub wykonać odpowiednie złącze przejściowe. Natomiast w drugim przypadku należy wykonać kabel z dwoma wtykami JACK, ale o polskim standardzie. Kable te, na pozór jednakowe, różnią się między sobą średnicą wtyku. W konsekwencji, gdy do współpracy z M 101 stosujemy oryginalny przewód połączeniowy, to nie odłącza on głośnika magnetofonu. Przez to zmienia się charakter obciążenia, co pociąga za sobą duże trudności przy ładowaniu programu z magnetofonu do pamięci komputera. Jeżeli oryginalny wtyk dociśnie się mocniej do gniazda głośnikowego M 101, to uzyska się odłączenie głośnika magnetofonu. Jednakże metoda ta jest niewygodna i mało skuteczna.

Zdarza się, że wpisany do ZX Spectrum program „zawiesi" się, tzn. nie reaguje na polecenia wydawane z klawiatury. Nie pozostaje wtedy nic innego, jak wyłączyć na chwilę zasilanie przez wyjęcie wtyczki z gniazda zasilającego komputer, kasując zawartość pamięci RAM. Częste korzystanie z tej metody może być przyczyną uszkodzenia gniazda zasilającego i powstania iskrzeń, które mogą spowodować zepsucie się mikrokomputera. Można temu zaradzić, gdyż na szynie krawędziowej ZX Spectrum jest wyprowadzona linia RESET. Podanie napięcia od 0 do 0,8 V na to wejście powoduje restart systemu. Można wykonać klawisz RESET (ZX Spectrum + ma taki klawisz). Należy użyć złącza krawędziowego 801 0620 1310021 lub 801 0960 1210021 (produkowanego przez UNITRA ELTRA w Bydgoszczy) oraz przełącznika, np. kontaktronowego, zwierającego linię RESET do masy (patrz dodatek F).

Częste dołączanie i odłączanie dodatkowych urządzeń zewnętrznych do szyny krawędziowej ZX Spectrum doprowadza do wytarcia się jej ścieżek, co w konsekwencji uniemożliwia współpracę z nimi. Aby tego uniknąć należy wykonać własną, dodatkową szynę krawędziową lub ograniczyć częstość dołączeń i odłączeń urządzeń do szyny wejścia—wyjścia ZX Spectrum.

* * Jak ZX Spectrum czyta klawiaturę? 1.7

Klawiatura ZX Spectrum składa się z 40 klawiszy. Jest ona podzielona na osiem części, po 5 klawiszy (każda). Każda z ośmiu części ma inny adres. Zaadresowanie daje możliwość sprawdzenia, czy któryś z 5 klawiszy podłączonych do wybranej części, został wciśnięty np. wybranie szóstego wiersza daje możliwość sprawdzenia klawiszy H, J, K, L i ENTER. Osiem wierszy klawiatury podłą-



Rys. 8. Klawiatura ZX Spectrum:
a — schemat, b — wnętrze (wygląd)

czone jest do szyny adresowej od linii A8 do linii A15 (wykorzystuje się tylko mniej znaczący bajt adresu) — rys. 8a. Adres danego wiersza jest generowany zgodnie ze wzorem:

$$\text{ADRES} = 254+256*(255-2^{7-n})$$

gdzie n jest numerem od 0 do 7.

Kolumny matrycy klawiszy (rys. 8b) są podłączone do szyny danych mikrokomputera przez układ ULA, np. wszystkie klawisze od V do B (V, G, T, 5, 6,

Y, H, B) są podłączone do linii D4, a klawisze od CAPS SHIFT do SPACE — do linii D0. Naciśnięcie dowolnego klawisza z danej kolumny powoduje, że na linii danych tej kolumny pojawia się niski stan napięcia — odpowiadający logicznemu zeru tylko wtedy, gdy wiersz, w którym znajduje się klawisz, był zaadresowany. Tak więc adres oraz odczytana informacja pozwalają na jednoznaczną interpretację, który klawisz został wciśnięty.

Komputer przeszukuje klawiaturę 50 razy na sekundę. Robi to w następujący sposób: generuje kolejno adresy wierszy klawiatury od pierwszego wiersza (32766) do ostatniego wiersza (65278) i sprawdza informacje na szynie danych. Wykrycie zera na jednym z bitów od D0 do D4 oznacza wciśnięcie określonego klawisza.

Sprawdzenie klawiatury odbywa się niezależnie od wykonywania innego programu, np. w Basicu. Stąd wniosek, że podczas wykonywania programu komputer niepotrzebnie traci czas na przeszukiwanie klawiatury. Przeszukiwanie klawiatury można wyłączyć wykonując instrukcję maszynową procesora DI (ang. *disable interrupt*), a włączyć instrukcją EI (ang. *enable interrupt*) — więcej informacji na ten temat można znaleźć w [4, 11].

Poniższy przykład pomoże zrozumieć zasady odczytywania klawiatury przez ZX Spectrum.

Wciśnięty zostaje klawisz K. Znajduje się on w siódmym wierszu. Jeżeli zostaje wygenerowany adres

$$254+256*(255-2^6) = 49\,150$$

to jest to równoznaczne z zaadresowaniem wiersza, w którym znajdują się znaki od H do ENTER. Jeżeli żaden klawisz nie byłby w tym wierszu przyciśnięty, to z układu wejścia—wyjścia o podanym adresie można odczytać wartość równą 191 (a nie 255 jak można by oczekiwać). Wynika to z faktu, że bit D6 szyny danych wykorzystywany jest do odczytu stanu gniazda EAR, współpracującego z magnetofonem. Jeżeli magnetofon nie jest podłączony, to przez cały czas ma on wartość logicznego zera (niski stan napięcia), wszystkie pozostałe bity mają stan wysoki — logiczna jedynka. Odczytana wartość jest więc równa $255-2^6 = 191$. Można to zobaczyć wykonując poniższy program

```
10 PRINT IN 49150:PAUSE 100
20 GO TO 10
```

W wyniku jego wykonania na ekranie będzie wyświetlona zawartość portu (układu wejścia—wyjścia) o adresie 49150. W omawianym przypadku wciśnięty jest klawisz K. Odczytana przez IN 49150 wartość będzie wynosiła 187. Wartości te zmieniają się zgodnie ze wzorem:

$$191-2^K \qquad K = 0, 1, 2, 3, 4$$

Wykładnik potęgi K odpowiada numerowi kolumny podłączonej przez ULA do odpowiedniego miejsca na szynie danych. Jeżeli zostanie wciśnięty inny klawisz z tego wiersza, np. H, to na ekranie zostanie wyświetlona liczba 175 = 191—16. Chcąc przekonać się o słuszności powyższych wywodów należy wykonać podany wyżej program, kilkakrotnie zmieniając adresy w instrukcji IN.

Nie są to wszystkie informacje dotyczące sposobu odczytywania klawiatury przez ZX Spectrum. Jak podano, klawiatura może pracować w jednym z trybów, np. L/C, G, E, K. System operacyjny, znajdujący się w pamięci stałej ROM, na podstawie stanu komórki MODE oraz FLAG (patrz dodatek C), informacji o tym, który klawisz został wciśnięty oraz czy był wcześniej wciśnięty klawisz SYMBOL SHIFT lub CAPS SHIFT, ustala ostatecznie jaka funkcja zostaje wybrana ze znaczeń danego klawisza.

* Odkrywając Basic 2

* Pierwszy krok 2.1

Użytkownik komunikuje się z mikrokomputerem ZX Spectrum za pośrednictwem języka Basic. ZX Spectrum, podobnie jak większość mikrokomputerów osobistych, ma interpreter języka Basic, umieszczony w pamięci stałej.

I n t e r p r e t e r jest programem napisanym w języku wewnętrznym procesora Z80. Analizuje on linia po linii program napisany w języku Basic i umieszczony w pamięci RAM. Rozpoznaje jakie instrukcje języka Basic zawiera linia, a następnie wykonuje odpowiednie obliczenia i inne konieczne operacje. W rzeczywistości komputer wykonuje jedynie program interpretera, a nie program w Basicu, choć użytkownik odnosi wrażenie, że wykonuje się właśnie jego program. Program w Basicu steruje pracą interpretera.

Opisana wyżej metoda realizacji języka symbolicznego, a więc niezrozumiałego dla komputera (językami takimi są wszystkie języki komputerowe wyższego rzędu, np.: Basic, FORTRAN, Pascal itd.), nosi nazwę i n t e r p r e t a c j i. Języki programowania realizuje się w komputerze także metodą kompilacji, w której program napisany w języku np. typu Pascal czy Basic, jest tłumaczony na równoważny mu funkcjonalnie program w języku wewnętrznym. Tłumaczenia dokonuje specjalny program nazywany k o m p i l a t o r e m (dokładniejsze wyjaśnienia w rozdz. 8).

Program interpretowany może być wykonywany bezpośrednio po wprowadzeniu go do pamięci, nie potrzeba czasu na tłumaczenie go na język wewnętrzny. Jest to zaletą interpretacji. Program interpretowany łatwiej uruchomić i testować niż program tłumaczony (kompilowany). Wadą interpretacji jest natomiast długi, w porównaniu z programem tłumaczonym, czas potrzebny na wykonanie programu. W gwarze informatycznej mówi się, że interpreter jest „wolny".

Programy umieszczone w pamięci stałej ZX Spectrum zajmują 16 kB. Oprócz tego, że interpretują one język Basic, obsługują także urządzenia wejścia—wyjścia. W pamięci stałej znajduje się także program, który pomaga wprowadzać

program w Basicu do pamięci za pomocą klawiatury, redagować go i poprawiać, tzw. edytor ekranowy. Edytor współpracuje z interpreterem. Każda wprowadzona do komputera linia programu jest analizowana. Tylko linia poprawnie napisana w języku Basic ZX Spectrum jest umieszczona w pamięci w formie odpowiedniej dla interpretera.

* Jak wygląda program w Basicu 2.2

Początki programowania LET, PRINT, REM 2.2.1

Program w języku Basic składa się z linii opatrzonych numerami. Linie mogą być wpisywane do pamięci w dowolnej kolejności, gdyż edytor automatycznie porządkuje linie w kolejności, wg rosnących numerów.

Oto prosty przykładowy program*) w Basicu, obliczający i drukujący kwadrat pewnej przypadkowo wybranej liczby całkowitej.

```
10 LET a=INT (100*RND)
20 LET a=a*a
30 PRINT a
40 REM to tylko komentarz
```

W programie tym w każdej linii znajduje się jedna instrukcja. W Basicu ZX Spectrum instrukcja rozpoczyna się tzw. słowem kluczowym. Są one wprowadzane z klawiatury za pomocą pojedynczego klawisza. Słowo kluczowe informuje interpreter jakie operacje należy wykonać.

Nauka Basica będzie ciekawsza i bardziej efektywna, gdy lekturę książki uzupełni się ćwiczeniami na komputerze. Dlatego proponujemy wykonywanie na Spectrum przykładowych programów. Jeżeli Czytelnik napotka trudności przy wpisywaniu programu do pamięci, może sięgnąć do punktu 2.3. Znajdzie tam potrzebne informacje.

Wpisany do pamięci program może być wykonany po wciśnięciu klawiszy:

RUN ENTER

Dokładniejszy opis sposobów uruchamiania programu znaleźć można w punkcie 2.4.

W podanym wyżej przykładzie, w linii 10**) jest losowo wybierana liczba. Wykonanie funkcji RND (zwanej generatorem liczb pseudolosowych) daje liczbę

*) Interpreter języka Basic utożsamia duże i małe litery w nazwach zmiennych, dlatego można ich używać zamiennie.

**) W podanym przykładzie numery kolejnych linii różnią się o 10. Mogą one zmieniać się z dowolnym krokiem.

losową (z przedziału (0, 1)). Liczba ta zostaje pomnożona przez 100 i zaokrąglona w dół do liczby naturalnej (za pomocą funkcji INT). W wyniku, w linii tej jest otrzymywana liczba losowa z przedziału (0,99). Generator liczb losowych oraz zasady wykonywania obliczeń zostały opisane dokładniej w punkcie 2.2.7.

Instrukcja LET jest tzw. *instrukcją podstawienia*. Zmienna, oznaczona tu literą a, przyjmie wartość obliczonego wyrażenia arytmetycznego znajdującego się po prawej stronie znaku =. W linii 10 zmienna a przyjmie wartość wybranej losowo liczby. W linii 20 zawartość zmiennej a zostanie zmieniona. Zmienna a przyjmie wartość równą kwadratowi jej poprzedniej wartości.

Instrukcja PRINT powoduje *drukowanie* zmiennej a na ekranie.

Po słowie kluczowym REM można napisać dowolny tekst, stanowiący *komentarz* do programu. Jest on umieszczony w programie, ale w czasie wykonywania interpreter ignoruje tekst komentarza (tekst znajdujący się po słowie REM aż do końca linii).

W języku Basic w jednej linii programu można umieścić kilka instrukcji. Omawiany program może być zapisany np. tak:

```
10 LET a = INT (100*RND):LET a = a*a:PRINT a
```

Jeżeli wprowadzi się do komputera linię programu, która nie ma numeru, to jego brak jest informacją dla interpretera, że linia powinna być wykonana natychmiast, a nie umieszczona w pamięci komputera. Taki sposób pracy nazywać będziemy *pracą w trybie natychmiastowym* (*bezpośrednim*). Linia napisana w trybie natychmiastowym może zawierać jedną lub kilka instrukcji. Jeśli napisze się:

```
LET a = INT(100*RND):LET a = a*a:PRINT a (ENTER)
```

to otrzyma się na ekranie wynik wykonania tej linii, lecz nie zostanie ona wprowadzona do pamięci. W trybie natychmiastowym ZX Spectrum może być używane jako kalkulator. Np. instrukcja:

```
PRINT 2*3+SIN (0.07)*49
```

wydrukuje obliczony wynik wyrażenia na ekranie.

Instrukcje pętli FOR ... NEXT oraz deklaracja tablicy DIM — 2.2.2

Instrukcja *pętli* ma następującą postać, np.:

```
10 FOR I = 1 TO 10
       ↑ ↑   ↑
       | |   końcowa wartość zmiennej I,
       | początkowa wartość zmiennej I,
       zmienna sterująca pętli (nazwa tej zmiennej musi być
       pojedynczą literą).
```

Instrukcja NEXT występuje zawsze w parze z instrukcją FOR i informuje interpreter, które instrukcje są zawarte wewnątrz pętli.

```
10 LET suma=0
20 FOR i=1 TO 10
30 LET suma=suma+i
40 NEXT i
50 PRINT suma
```

W przykładowym programie wewnątrz pętli, tzn. między instrukcjami FOR i NEXT, znajduje się linia 30. Instrukcje zawarte wewnątrz pętli, w tym przypadku LET SUM = SUM+I, wykonuje się dziesięciokrotnie, przy czym wartość zmiennej I będzie wynosiła wewnątrz pętli kolejno 1, 2, 3, ..., 10.

Zmienna sterująca pętli będzie zmieniała się z krokiem 1 o czym nie trzeba informować interpretera. Wartość kroku, z jakim zmienia się zmienna sterująca I po każdym wykonaniu pętli, może być różna od jedności. Jeśli pętla z przykładowego programu miałaby się wykonać dla kolejnych wartości zmiennej I = 0, 3, 6, 9, instrukcja z linii 20 powinna mieć postać:

20 FOR I = 0 TO 9 STEP 3

Słowo kluczowe STEP informuje interpreter o k r o k u, z którym należy zmieniać wartość zmiennej sterującej. Krok może być liczbą ujemną. Wartości: początkowa i końcowa zmiennej sterującej oraz krok mogą być ułamkowe. W Basicu ZX Spectrum poprawna jest np. taka instrukcja:

FOR I = 0.5 TO 1.3 STEP 0.1

Uwaga! *Po zakończeniu wykonania instrukcji pętli zmienna sterująca przyjmie wartość końcową plus krok.*

Instrukcje pętli mogą być zawarte jedna w drugiej. Poniżej podany program wypełnia jedynkami tablicę:

```
 5 DIM a(5,5)
10 FOR i=1 TO 5
20 FOR j=1 TO 5
30 LET a(i,j)=1
40 NEXT j
50 NEXT i
```

pętla wewnętrzna (linie 20–40); pętla zewnętrzna (linie 10–50)

Można umieścić wiele pętli jedna w drugiej. „Granice" pętli nie powinny się jednak przecinać. Podany niżej program, w którym granice pętli przecinają się, jest niepoprawny, choć interpreter nie zawsze wskaże błąd.

```
 5 DIM a(5,5)
10 FOR i=1 TO 5
20 FOR j=1 TO 5
30 LET a(i,j)=1
40 NEXT i
50 NEXT j
```

W linii 5 oraz w linii 30 pojawiły się nowe instrukcje języka. W języku Basic zmienne mogą być zgrupowane w tzw. t a b l i c ę, którą można porównać z wektorem lub macierzą liczbową. Tablica może być wielowymiarowa — w przykładowym programie tablica ma dwa wymiary i zawiera pięć wierszy i pięć kolumn (inaczej mówiąc pięć wierszy po pięć elementów w każdym, czyli 25 elementów). Zanim tablica zostanie użyta, trzeba określić jej rozmiary. Do tego celu służy instrukcja DIM (z ang. *DIMension*). Wymiary tablicy potrzebne są interpreterowi, by mógł zarezerwować dla niej miejsca w pamięci komputera.

Uważny Czytelnik z pewnością zauważy, że nie ma potrzeby wcześniejszego deklarowania takich zmiennych jak np. zmienna I w przykładowych programach. Zmienne takie nazywamy z m i e n n y m i p r o s t y m i, dla odróżnienia od zmiennych, które znajdują się w tablicy — nazywanych z m i e n n y m i s t r u k t u r a l n y m i. Zmienne te pamiętają zarówno liczby całkowite, np. 7, −2, jak i tzw. liczby z m i e n n o p r z e c i n k o w e*), tzn. mające kropkę dziesiętną — np. 7.52 lub −0.02 itp. Interpreter przydziela zmiennym prostym miejsce w pamięci wtedy, gdy zmiennej takiej zostanie po raz pierwszy przypisana jakaś wartość, np. w instrukcji podstawiania (instrukcja LET), bowiem wielkość pamięci potrzebnej do jej zapamiętania jest stała (w ZX Spectrum wynosi 5 bajtów).

Inaczej rzecz się ma dla zmiennych strukturalnych. Odwołanie do takiej zmiennej (patrz linia 30: A(1, J)), nie zawiera informacji o wielkości tablicy i stąd konieczność by rozmiar tablicy był z góry znany.

Elementy tablicy mogą występować jako argumenty obliczeń, relacji logicznych itp. Do elementów tablicy można odwołać się podając nazwę tablicy i numery elementów kolejnych wymiarów. Na przykład, odwołanie do piątego elementu trzeciego wiersza tablicy zadeklarowanej: DIM A(5,5) ma postać A(3,5). Odwołanie do trzeciego elementu wektora: DIM x(3) jest następujące: x(3).

Nieprzypadkowo tablice zostały opisane przy okazji instrukcji FOR. Instrukcja ta jest bowiem najczęściej używana, gdy wykonuje się operacje na elementach tablicy, lub ujmując rzecz szerzej; na sekwencji obiektów (np. liczb), które można ponumerować.

Uwaga! *Nazwa tablicy może składać się tylko z jednej litery.*

Instrukcja warunkowa IF, instrukcja skoku GO TO 2.2.3

Często zdarza się, że należy zaprogramować „podjęcie decyzji" typu: jeżeli coś będzie prawdą, wykonaj pewną czynność, jeśli będzie fałszem — nie. Język Basic ma do tego celu odpowiednią instrukcję:

IF *warunek* THEN *instrukcja*

Instrukcje, jedna lub kilka, następujące po słowie kluczowym THEN aż do

*) Są one zapamiętywane w postaci wykładniczej, np. $0.752 \cdot 10^{-1}$ dla liczby 7.52.

końca linii, wykonują się wtedy tylko, gdy warunek będzie prawdziwy. Po słowie kluczowym THEN można umieścić dowolne instrukcje Basica, np. następną instrukcję IF.

W podanym niżej programie w linii 30 wybiera się losowo liczbę z przedziału (−5; 4). W liniach 40 i 45 oblicza się i drukuje wartość bezwzględną liczby.

```
30 LET a=INT (RND*10)-5: PRINT
a,
40 IF a<0 THEN LET a=-a
45 PRINT a
```

Jak wygląda warunek? Może on zawierać jedną z następujących relacji:

= — równe,
> — większe,
< — mniejsze,
>= — większe lub równe,
<= — mniejsze lub równe,
<> — różne.

W relacjach tych porównuje się ze sobą dwie zmienne lub zmienną z konkretną liczbą, tzw. stałą.

Relacja po obliczeniu uzyskuje wartość logiczną 1 (prawda) lub 0 (fałsz). Na obliczonych w taki sposób wartościach można przeprowadzać dalsze operacje za pomocą operatorów logicznych: AND, OR lub NOT*).

Operatory NOT (negacja), AND (iloczyn logiczny, koniunkcja) oraz OR (suma logiczna, alternatywna) w ZX Spectrum działają trochę inaczej niż ma to miejsce w matematyce. Operator NOT (negacja) zmienia wartość logiczną wyrażenia na przeciwną: NOT x daje wartość 0 (fałsz), gdy x ma wartość różną od 0 oraz wartość 1, gdy x ma wartość 0. Wyrażenie x AND y daje wartość 0 (fałsz), jeżeli y ma wartość 0 (fałsz), oraz wartość x, gdy y ma wartość różną od zera — wynik nie zależy wtedy od wartości x (x może być tu zmienną tekstową lub łańcuchem znaków; jeśli y = 0 to wynikiem relacji jest tekst pusty). Wyrażenie x OR y daje wartość 1 (prawda), gdy y ma wartość różną od zera, oraz wartość x (niezależnie od tego, jaką wartość ma x), gdy y ma wartość 0 (fałsz).

Kolejność wykonywania operatorów jest następująca: najpierw operator NOT, następnie AND, a na końcu OR. Dlatego niekiedy trzeba stosować nawiasy, by wyrażenie obliczone było w odpowiedniej kolejności (podobnie jak to ma miejsce w matematyce, gdy np. oblicza się wartość wyrażenia, w którym występują dodawanie i mnożenie).

Oto przykład skomplikowanego, lecz poprawnego wyrażenia w języku Basic:

IF (A = 2) AND (B < 0 OR B > 0.7) AND NOT (B = 5) THEN...

Wykonanie instrukcji GO TO *numer linii* powoduje, że interpreter będzie wykonywał dalej program, od linii o podanym numerze (działanie instrukcji

*) Operatory logiczne mają mniejszy priorytet niż relacje, np. NOT A = B jest równoważne NOT (A = B).

GO TO*) jest adekwatne do jej nazwy — z ang. *GO TO* — idź do). Instrukcja GO TO nazywana bywa często instrukcją skoku bezwarunkowego.

Instrukcji GO TO używa się zazwyczaj razem z instrukcją IF. Używanie instrukcji GO TO tylko po to, by zmienić kolejność wykonywania instrukcji jest złą zasadą programowania. Podany niżej przykład ilustruje na czym polega owa zła, choć z punktu widzenia interpretera Basica, poprawna zasada:

```
20 GO TO 50
30 IF a<0 THEN LET a=-a
40 GO TO 70
45 PRINT a
50 LET a=INT (RND*10)-5: PRINT
a,
60 GO TO 30
70 PRINT a
```

Program napisany w taki sposób trudno przeanalizować i zrozumieć. Często autor programu, po pewnym czasie, nie może zrozumieć swego dzieła. Dlatego taki sposób programowania stanowczo się odradza.

Natomiast instrukcja GO TO używana po instrukcji IF jest wielce przydatna. Można za jej pomocą zorganizować pętle innego typu niż pętla FOR, np. pętlę, która wykonuje się, gdy spełniony jest pewien warunek. Jeżeli warunek ten przestaje być prawdziwy, to wykonywanie pętli kończy się.

Podany niżej program normalizuje losową liczbę z zakresu od 1 do 999 (linia 10) do postaci wykładniczej, np. liczba 300 zamieniana jest na liczbę $0.3*10^3$ (co będzie zapisane 10E3).

```
10 LET x=INT (999*RND)+1
20 LET ex=0
30 LET x=x/10
40 LET ex=ex+1
50 IF x>1 THEN GO TO 30
60 PRINT x;"*10e";ex
```

W linii 60 drukuje się znormalizowaną już liczbę. Kolejne elementy, które mają być wydrukowane (zmienna x, tekst "*10E" oraz zmienna Ex) oddzielone zostały znakiem średnika. Znak ten informuje interpreter, że mają one być wydrukowane w jednej linii, jeden za drugim, bez odstępów (spacji) między nimi.

Co się stanie, jeżeli w linii 50 napisze się: GO TO 25, zamiast 30, a więc zaprogramuje się skok do linii, której nie ma w programie? Interpreter ZX Spectrum jest w takim przypadku wyrozumiały. Wykona skok do linii następnej — w tym przypadku następną, po nieistniejącej linii 25, byłaby 30.

Instrukcji GO TO można użyć także w inny sposób. Mianowicie numer linii, do której ma nastąpić skok, może być wyliczony w czasie wykonywania programu (taka postać funkcji przypomina instrukcję ON...GO TO z języka FORTRAN lub CASE z języka Pascal), np.:

*) Instrukcja GO TO na klawiszu G jest oznaczona GOTO.

```
  10 LET a=10
  20 IF RND>0.5 THEN LET a=20
  30 GO TO 50+a
  60 PRINT "to jest linia 60": G
O TO 80
  70 PRINT "to jest linia 70"
  80 PRINT "koniec"
```

Jeżeli we fragmencie programu z linii 10 i 20 zmienna A przyjmie wartość równą dziesięć, nastąpi skok do linii 60, jeśli dwadzieścia — do linii nr 70.

Instrukcją skoku wyliczonego można zastąpić kilka instrukcji IF, jak ma to miejsce w podanym niżej przykładzie.

Fragment programu*)

```
.
.
.
50 IF A < 0 THEN GO TO 100
60 IF A = 0 THEN GO TO 150
70 IF A > 0 THEN GO TO 200
.
.
.
```

można zapisać:

```
.
.
.
50 GO TO (A < 0)*100+(A = 0)*150+(A > 0)*200
.
.
.
```

Relacja postaci A < 0, A = 0 przyjmuje wartość 0 (fałsz) lub 1 (prawda). Tylko jedna z trzech relacji znajdujących się w linii 50 jest w danej chwili prawdziwa i dlatego nastąpi skok do linii 100, 150, 200, w zależności od wartości A.

Bezpośredni dostęp do pamięci RAM oraz urządzeń zewnętrznych: PEEK, POKE, IN, OUT 2.2.4

Za pomocą instrukcji POKE można liczbę z zakresu od 0 do 255 zapisać bezpośrednio w konkretnej komórce (pojedynczym bajcie pamięci). Instrukcja POKE ma postać:

*) Kropeczkami zastąpiono brakujący fragment programu, w którym m.in. zmienna A przyjmuje pewną wartość.

POKE *adres*, *liczba*

```
POKE adres, liczba
      ↑      ↑
      |      wartość, która ma być
      |      wpisana (0 do 255)
      adres miejsca pamięci,
      do którego należy
      wpisać (od 0 do 65535)
```

Na przykład POKE 50120, 250 oznacza: zapisz w komórce pamięci o adresie 50120 liczbę 250.

Uwaga! *Interpreter nie sprawdza czy adres podany w instrukcji POKE jest adresem pamięci RAM (tzn. od 16384 do 65535).*

Instrukcja OUT przypomina instrukcję POKE. Zapis następuje jednak do urządzeń we/wy*).

Dowolną komórkę (bajt) pamięci (RAM oraz ROM) można odczytać za pomocą funkcji PEEK. Na przykład:

10 PRINT PEEK 20325

spowoduje, że wartość komórki pamięci o adresie 20325 zostanie wydrukowana. PEEK jest funkcją, która nie może wystąpić sama, jako instrukcja. Może wystąpić jedynie z instrukcją LET, PRINT, itd.

Funkcja IN odczytuje bajt z urządzenia zewnętrznego**) o podanym adresie np.:

```
10 LET A = IN 23513
              ↑
              adres układu wejścia — wyjścia
```

Za pomocą IN i OUT można w Basicu współpracować z urządzeniami zewnętrznvmi.

Wprowadzanie danych do programu: INPUT, READ, DATA, RESTORE, INKEYS 2.2.5

Jak należy postąpić jeżeli chce się wprowadzić dane z klawiatury do komputera? Można w tym celu użyć instrukcji INPUT.

10 INPUT A
20 PRINT A

Po wykonaniu instrukcji INPUT (linia 10) interpreter oczekuje na daną. W ostatniej linii ekranu wyświetlany jest migający kursor. Należy wprowadzić potrzebną

*) Zapisu informacji do urządzenia we/wy interpreter dokonuje za pomocą operacji języka maszynowego out (c),a.

**) Interpreter wykonuje operację maszynową in a,(c).

daną z klawiatury, wciskając po zakończeniu operacji klawisz ENTER, np. 123 ENTER. Na ekranie wyświetli się wartość zmiennej A.

Pojedynczą instrukcją INPUT można wczytać wartości kilku zmiennych, np.:

10 INPUT A, B

Po wykonaniu tej instrukcji komputer oczekuje na wartość pierwszej zmiennej. Gdy się ją wprowadzi i wciśnie klawisz ENTER, oczekuje na wartość drugiej.

Oto następny krótki program, używający instrukcji INPUT:

```
  10 INPUT "Podaj dzisiejsza dat
e ";a;b;c
  20 PRINT "dzien:";a
  30 PRINT "miesiac:";b
  40 PRINT "rok:";c
```

W wyniku wykonania instrukcji z linii 10, w dolnej części ekranu pojawia się tekst:

Podaj dzisiejszą datę [L]
migający kursor ↑

Należy wtedy wprowadzić wartości, które mają być nadane zmiennym A, B, C. Instrukcja INPUT pozwala nie tylko wczytywać dane, ale także drukować informacje na ekranie. W programie podanym wyżej w instrukcji tej drukowano tekst. Można także wydrukować wartości zmiennych (podobnie jak w instrukcji PRINT). Nazwy zmiennych muszą być umieszczone w nawiasach, np.:

```
  10 LET a=INT (10*RND)
  20 INPUT "Ile wynosi 10*";(a);
"?",b
  30 IF 10*a<>b THEN GO TO 20
  40 PRINT "ok"
```

Linia 20 zapisana niżej jest dla interpretera równoważna linii 20 przykładowego programu:

20 INPUT ("Ile wynosi 10*"; A); B

średnik oznacza, że wartość zmiennej zostanie wydrukowana bezpośrednio (bez odstępu) po poprzednio wydrukowanym tekście.

Instrukcja INPUT drukuje informacje i wprowadzane znaki w tzw. o k n i e s y s t e m o w y m, czyli dwu ostatnich liniach ekranu. Linie te numerowane są w ten sposób, że linia przedostatnia ma numer 0, linia ostatnia — 1. Po wykonaniu instrukcji INPUT okno systemowe jest kasowane.

W instrukcji INPUT można także używać o p c j i*) AT. Podaje się po niej

*) Opcja — możliwość do wyboru. W tym przypadku rozumie się przez to określenie część instrukcji, która może wystąpić, ale nie jest konieczna.

numery wiersza i kolumny, od których należy rozpocząć drukowanie (wczytywanie), np.:

```
INPUT AT 0, 0
         ↑  ↑
         |  kolumna
       wiersz
```

Jeżeli w instrukcji INPUT AT poda się numer wiersza równy 0 lub 1, to informacje wprowadzane są w przedostatnim (zero) lub ostatnim (jeden) wierszu ekranu. Jeśli jednak numer wiersza jest większy od liczby jeden, to interpreter przesuwa tekst wydrukowany na ekranie w górę o tyle wierszy, ile potrzeba, by wiersz o podanym numerze znalazł się w ostatniej linii ekranu (linia 0 okna systemowego przesuwa się w górę) lub kasuje część ekranu (patrz p. 3.4.1). Takie przesuwanie tekstu nazywa się skrolowaniem (z ang. *scroll* znaczy przewiń).

Jak napisano wyżej informacje mogą być wczytywane w dwu ostatnich liniach ekranu. Mogą być wczytywane także w dolnej linii — pod warunkiem, że wcześniej okno systemowe zostanie ,,powiększone" za pomocą instrukcji INPUT AT, np.:

```
10 INPUT AT 10,0; "To jest linia nr 10"; AT 2,0; "To jest linia
                 ↑                                        nr 2"; A
powiększenie                                                ↑
okna systemowego                                            |
```

ten tekst wydrukuje się w drugiej linii okna systemowego, znajdującej się w dziesiątym wierszu ekranu, licząc od dołu

Operacje ,,powiększania" okna oraz wprowadzania informacji muszą być umieszczone w jednej instrukcji INPUT, bowiem po zakończeniu wykonania tej instrukcji interpreter kasuje okno systemowe, a następnie ,,zmniejsza" je do dwu ostatnich linii ekranu.

Instrukcja INPUT pozwala również na wprowadzanie tekstów do programu. Tekst taki może być zapamiętany w zmiennej tekstowej. Dotychczas zostały omówione zmienne liczbowe. Natomiast zmienna tekstowa może pamiętać cały ciąg znaków o długości od 0 (mówimy wtedy, że zmienna zawiera tekst pusty) aż do wyczerpania wolnych komórek pamięci RAM. Tekst może być złożony z dowolnych znaków: cyfr, liter dużych i małych, znaków takich, jak dwukropek, nawias, itd. Zmienne tekstowe muszą mieć nazwę złożoną z pojedynczej litery oraz znaku $, np. a$, z$, A$.

Podany niżej program wczytuje tekst do pamięci komputera, zapamiętuje pod zmienną a$ i drukuje na ekranie:

```
10 INPUT a$
20 PRINT a$
```

W wyniku wykonania instrukcji z linii 10 interpreter oczekuje ciągu znaków,

zakończonego znakiem ENTER (jeśli naciśniemy tylko ENTER, zmienna a$ zawierać będzie tekst pusty). Migający kursor znajduje się w takim przypadku między znakami cudzysłowu: ,,[L]"

Uwaga! *Jeżeli napisze się w programie INPUT LINE a$, zamiast INPUT a$, to interpreter nie wyświetli znaków cudzysłowu.*

Instrukcja INPUT pozwala na wprowadzenie danych do programu za pomocą klawiatury. Dane można wprowadzać także ze specjalnego zbioru: DATA. W zbiorze tym umieszcza się takie dane, które są potrzebne przy każdym wykonaniu programu. Zbiór, o którym mowa, tworzy się instrukcją DATA. Po słowie kluczowym DATA podaje się jeden lub kilka elementów oddzielonych przecinkami. Kolejne wartości zapamiętane w tym zbiorze mogą być odczytywane specjalną instrukcją READ. Przypomina ona INPUT, ale służy wyłącznie do odczytywania zbioru DATA, np.:

```
10 DATA 10,20,30
20 READ a
30 READ b,c
40 PRINT a,b,c
```

Instrukcja DATA może być umieszczona w programie w dowolnym miejscu, bowiem interpreter w czasie wprowadzania programu do pamięci tworzy sobie zbiór DATA, w którym ,,składa" razem wszystkie wartości umieszczone w instrukcjach DATA w całym programie, np.:

```
10 FOR i=1 TO 4
20 DATA 1,2,3
30 READ a
40 PRINT a
50 NEXT i
60 DATA 4
```

Po wykonaniu powyższego programu na ekranie wydrukowane będą liczby 1, 2, 3, 4. Chociaż linie o numerach 20, 30, 40 zostaną wykonane cztery razy w pętli, to w czasie wykonywania programu są ignorowane. W podanym wyżej programie w zbiorze DATA znajdą się liczby: 1, 2, 3 i 4. Jeżeli w czasie wykonywania programu wszystkie liczby ze zbioru DATA zostały już odczytane i wykonuje się następną instrukcję READ, to interpreter sygnalizuje błąd: OUT of DATA. Taka sytuacja nastąpiłaby, gdyby w programie zabrakło np. linii 60.

Zbiór DATA można ,,przewinąć" na początek za pomocą instrukcji RESTORE, np.:

```
10 DATA 1,2,3
20 READ a,b,c
30 PRINT a;b;c
40 RESTORE
45 PRINT
50 READ x,y,z
60 PRINT a;b;c
```

Interpreter w czasie wprowadzania programu utworzy zbiór DATA zawierający liczby 1, 2, 3. Po wykonaniu instrukcji z linii 40 dane będą odczytywane instrukcją READ od początku (od liczby 1). Po wykonaniu programu na ekranie zobaczymy liczby 1, 2, 3, 1, 2, 3.

Można także „przewinąć" część zbioru DATA. Dzieje się tak w poniższym programie:

```
10 DATA 5,10
20 READ a,b,c
30 DATA 20,30
40 RESTORE 30
45 PRINT
50 READ x,y
60 READ v,z
70 DATA 50,60
80 PRINT a,b,c,x,y,v,z
```

Zbiór DATA po wprowadzeniu programu zawiera liczby:
5, 10, 20, 30, 50, 60. Po wykonaniu linii 20, z tego zbioru można było przeczytać liczby: 30, 50, 60 (5, 10, 20 zostały już przeczytane).

Instrukcja RESTORE 30 „przewija" plik DATA w ten sposób, że do czytania są dostępne dane z instrukcji DATA, znajdujące się w programie począwszy od tej linii do końca, a więc w tym przypadku liczby 20, 30, 50, 60 (z programu począwszy od linii 30 do końca).

W zbiorze DATA można umieścić także teksty, np.:

```
10 DATA "ala","ela","ola"
20 READ a$,b$,c$
30 PRINT a$,b$,c$
```

Można także „mieszać" tekst i liczby. Trzeba wówczas uważać: np. gdy ze zbioru DATA chce się czytać tekst, powinien znajdować się tam tekst, a nie liczba i odwrotnie. W przeciwnym bowiem razie interpreter zasygnalizuje błąd. Na przykład, poprawnie wykona się program:

```
10 DATA "ala",7
20 READ a$,x
30 PRINT a$,x
```

ale niepoprawny jest program, w którym linię 20 zmieni się na:

```
20 READ x, a$
```

Do wprowadzenia danych służy także funkcja INKEY$. Funkcja ta odczytuje klawiaturę i podaje w wyniku znak (w trybie L lub C zależnie od stanu klawiatury) odpowiadający wciśniętemu klawiszowi lub tekst pusty, gdy żaden nie jest przyciśnięty. Funkcja INKEY$ w odróżnieniu od funkcji INPUT nie czeka na naciśnięcie klawisza! Należy więc używać funkcji INKEY$, np. w taki sposób:

```
  10 IF INKEY$="" THEN GO TO 10:
 REM oczekiwanie na wcisniecie k
lawisza
  20 PRINT INKEY$
  30 IF INKEY$<>"" THEN GO TO 30
: REM oczekiwanie na puszczenie
klawisza
  40 GO TO 10
```

(Program można zatrzymać wciskając Symbol Shift oraz BREAK)

Podprogramy GO SUB, RETURN, definiowanie funkcji DEF FN, FN 2.2.6

Często pewien fragment powtarza się kilkakrotnie w programie. Powtarzające się linie mogą być zapisane jeden raz jako tzw. p o d p r o g r a m. Podprogram ten jest wywoływany wtedy, gdy chcemy ów wydzielony fragment wykonać. Podprogram wywołuje się za pomocą instrukcji GO SUB. Przypomina ona skok GO TO. Jednak w tym przypadku interpreter zapamiętuje miejsce wywołania i po zakończeniu wykonania podprogramu „wraca" do tego miejsca.

Wywołanie podprogramu ma postać:

GO SUB *numer*

↑ numer linii, od której rozpoczyna się podprogram (może być również wyrażenie)

W Basicu ZX Spectrum nie ma żadnej specjalnej instrukcji wyznaczającej początek podprogramu. Podprogram nie różni się niczym od programu. Jest to sekwencja ponumerowanych linii, z jedną różnicą: na końcu podprogramu znajduje się instrukcja RETURN, stanowiąca informację dla interpretera, że należy „powrócić" do wykonywania programu głównego. Podprogram może oczywiście wywoływać kolejny podprogram.

Jeżeli zostanie wykonana instrukcja RETURN, choć nie było wcześnie instrukcji GOSUB, interpreter sygnalizuje błąd — RETURN without GOSUB.

Oto przykład programu zawierającego podprogram zaczynający się od linii 50. Podprogram ten „zeruje" górne 10 linii ekranu, drukując tam znak odstępu (tzw. spacji) 32 razy w każdym wierszu (patrz str. 50).

W linii 60 znajduje się instrukcja STOP. Zgodnie ze swą nazwą powoduje ona zatrzymanie wykonania programu oraz wydrukowanie w dolnej części ekranu komunikatu: STOP statement, *nr linii*. Gdyby nie było linii 60, wykonałby się podprogram od linii 500 i nastąpiłby błąd: RETURN without GOSUB.

```
  20 FOR i=1 TO 22
  30 PRINT "*********************
***********"
  40 NEXT i
  50 GO SUB 500: REM zerowanie e
kranu
  60 STOP
 499 REM poczatek podprogramu
 500 PRINT AT 0,0;
 510 FOR j=1 TO 10
 520 PRINT "
                    "
 530 NEXT j
 540 RETURN
```

W linii 500 znajduje się „dziwna" instrukcja PRINT. Po słowie kluczowym AT podano miejsce na ekranie, w którym należy rozpocząć drukowanie, np.:

```
PRINT AT 0, 0
         ↑  ↑
         |  numer znaku w wierszu (0÷31)
         numer wiersza (0÷21)
```

W przykładowym programie drukowanie rozpoczyna się w lewym górnym rogu. Gdyby w omawianym programie zabrakło linii 500, to podprogram zerowałby dziesięć kolejnych linii ekranu, począwszy od następnej po tej, w której ostatnio drukowano.

Niekiedy istnieje potrzeba przekazywania danych do podprogramu, tzw. parametrów. Przykładem może być podprogram obliczający funkcję sin *x*, gdzie *x* jest parametrem. W Basicu ZX Spectrum nie ma mechanizmów zapewniających „automatyczne" przekazywanie parametrów do podprogramów. Można je przekazywać jednak za pomocą zmiennej (zmiennych) lub określonych miejsc (komórek) pamięci. Jeżeli omawiany program miałby zerować nie 10, ale K linii ekranu, to informację o liczbie linii, które należy wyzerować, można przekazać za pośrednictwem zmiennej (nazwanej tu umownie, dla ułatwienia, literą K). Program może być napisany np. tak:

```
  20 FOR i=1 TO 22
  30 PRINT "*********************
***********"
  40 NEXT i
  50 INPUT "ile linii zerowac?";
k
  60 GO SUB 500: REM zerowanie e
kranu
  70 STOP
 499 REM poczatek podprogramu
 500 IF k=0 THEN RETURN
 510 PRINT AT 0,0;"
                    "
 520 IF k=1 THEN RETURN
 530 FOR j=2 TO k
 540 PRINT "
                    "
 550 NEXT j
 560 RETURN
```

Podany program jest zabezpieczony na wypadek wprowadzenia złej (np. ujemnej) wartości parametru K. Podprogram może zawierać kilka instrukcji RETURN. Instrukcja RETURN nie musi być ostatnią „napisaną" instrukcją podprogramu.

Parametry można także przekazywać za pośrednictwem określonych komórek pamięci*⁾. Niżej podano wersję programu zerowania K linii ekranu, w której informacja o liczbie linii do wyzerowania jest przekazywana za pośrednictwem komórki pamięci o adresie 23296 (pierwszy bajt nieużywanego w tym programie bufora drukarki):

```
  20 FOR i=1 TO 22
  30 PRINT "**********************
************"
  40 NEXT i
  50 INPUT "ile linii zerowac?";
l
  55 POKE 23296,l
  60 GO SUB 500: REM zerowanie e
kranu
  70 STOP
 499 REM poczatek podprogramu
 500 LET k=PEEK 23296
 505 IF k=0 THEN RETURN
 510 PRINT AT 0,0;"
                                "
 520 IF k=1 THEN RETURN
 530 FOR j=2 TO k
 540 PRINT "
           "
 550 NEXT j
 560 RETURN
```

Podprogramy należy pisać nie tylko wtedy, gdy jakaś część programu powtarza się kilkakrotnie. Pewne fragmenty programu można zapisać w postaci podprogramu nawet wtedy, gdy podprogram taki zostanie wywołany tylko jeden raz. Program tak napisany ma bardziej przejrzystą strukturę, łatwiej go zarówno pisać, jak i analizować. Zdarza się niekiedy, że program napisany w języku Basic komputera pewnego typu, trzeba powtórnie napisać (jak to się mówi w gwarze informatycznej — przenieść) na inny komputer. Języki Basic różnych komputerów różnią się m.in. instrukcjami graficznymi. Dlatego warto te fragmenty programu, w którym używa się unikalnych dla danego typu komputera instrukcji zapisać jako podprogramy. Przy przenoszeniu programu wystarczy zmienić owe podprogramy, zamiast analizować cały program i poprawiać go.

*⁾ Jest to szczególnie przydatne, gdy program w Basicu łączy się z programami np. w języku asemblera. W Basicu tego sposobu nie poleca się, choć dla ilustracji zasady podano program.

W programach napisanych w języku Basic ZX Spectrum można wykonywać obliczenia arytmetyczne np.:

```
10 LET A = 2+c*3.42/d1-0.24
```

albo

```
10 PRINT 3*ab — 7.1/4
```

W obliczeniach można używać stałych (np. 2; 3.42; −0.24) oraz zmiennych (A; d1; ab). Zmiennym tym należy nadać wartość za pomocą instrukcji INPUT, LET lub READ zanim zostaną użyte w wyrażeniu. Zmienna musi mieć nazwę zaczynającą się od litery (dużej lub małej). W nazwie można umieścić litery, cyfry oraz znak odstępu (spację), np.: A10, ZX81, Alabama, itd. Interpreter nazwy zapisane dużymi i małymi literami traktuje jako takie same. Na przykład takimi samymi nazwami są:

```
ala
ALA
a la
AL A
```

Spacje umieszczone w środku nazwy nie zmieniają jej, służą wyłącznie wygodzie użytkownika.

Liczby, oprócz takiej formy jak podana wyżej, mogą być zapisywane w postaci wykładniczej, np.: 5E17 lub 5e17 co oznacza $5 \cdot 10^{17}$.

Komputer ma ograniczoną ilość miejsca, w której może być pamiętana liczba. Każdą liczbę zapamiętuje się na 5 bajtach. Dlatego bardzo duże liczby są pamiętane w „zaokrągleniu" (wyjaśnienie jak zapamiętuje się liczby można znaleźć w p. 2.5). Prowadzi to do następujących paradoksów. Na przykład, jeśli napiszemy:

```
PRINT 1e10+1-1e10
```

to otrzymamy wynik 0 zamiast 1, bowiem $10^{10}+1$ to dla komputera nadal 10^{10}
Poprawny wynik otrzymamy pisząc:

```
PRINT 1e10-1e10+1
          =0    +1 = 1
```

Jeżeli zmienna użyta w wyrażeniu arytmetycznym nie ma nadanej wcześniej wartości, to interpreter zasygnalizuje błąd: Variable not found.

```
10 LET A = B*30/100
```

Jeżeli zmienna B nie otrzymała wartości (np. w instrukcji LET, INPUT), to wystąpi błąd.

W obliczeniach realizowanych na ZX Spectrum można używać wielu funkcji matematycznych (mają one mniejszy priorytet niż operatory):

- ↑ oznacza potęgowanie, np. 2 ↑ 5 oznacza $2^5 = 2*2*2*2*2$
- SQR — pierwiastek kwadratowy, np. SQR 4 oznacza $\sqrt{4}$, SQR a oznacza $\sqrt{a}$.
- ABS — wartość bezwzględna, np. ABS a oznacza |a|, ABS (−2) równe jest 2, a ABS 2 równe jest 2.
- EXP x = e^x
- LN x = ln(x), LNx oblicza wartość logarytmu naturalnego*).
- SIN x = sin(x)
- COS x = cos(x)
- TAN x = tg(x)
- ASN x = arcus sin(x)
- ACS x = arcus cos(x)
- ATN x = arcus tg(x)
- INT X oblicza część całkowitą liczb zaokrąglając ją w dół, np. INT. 3.14 równa się 3, a INT (−3,14) = −4.

Interpreter pamięta także wartość liczby π(PI). Można takiej liczby użyć w wyrażeniach arytmetycznych (oznaczenie PI znajduje się na klawiszu M).

W programach można używać także własnych funkcji. Funkcje te należy zawczasu zdefiniować za pomocą polecenia DEF FN:

```
10 DEF FN L(a, x) = LN x/LN a
          ↑ ↑
          |  parametry
          nazwa funkcji (jedna litera)
```

Gdy funkcja jest bezparametrowa, to po nazwie funkcji należy podać jedynie nawiasy, np.: DEF FN a() ... Nazwy parametrów muszą być jednoliterowe. Parametrem może być także zmienna tekstowa i wtedy nazwa składa się z litery i znaku $. Po znaku = piszemy treść funkcji, którą jest dowolne wyrażenie języka Basic. Mogą się tam znaleźć wywołania zdefiniowanych wcześniej funkcji. W treści funkcji można oczywiście używać dowolnych zmiennych, a nie tylko parametrów, jak to ma miejsce w podanej tu jako przykład funkcji obliczającej $\log_a(x)$.

Oto przykład wywołania zdefiniowanej wyżej funkcji:

```
20 PRINT FN L (2,3)
```

albo

```
20 LET A = 5
30 LET B = FN L(2, A): PRINT B
```

albo

```
20 LET B = FN L(2, SQR (SIN PI/2)): PRINT B
```

Słowo kluczowe FN informuje interpreter, że nastąpiło wywołanie funkcji (w nawiasach podaje się wartości parametrów, dla których funkcja ma być obliczona).

*) ZX Spectrum nie ma funkcji obliczania logarytmu o dowolnej podstawie, ale wiadomo, że $\log_a(x) = \ln(x)/\ln(a)$, więc logarytm taki można obliczyć pisząc: LN(x)/LN(a).

Uważny Czytelnik spostrzegł, że wywołanie funkcji nie może samo znaleźć się w linii programu. Nie można napisać np. takiej linii:

```
10 SIN(A)
```

Interpreter musi gdzieś umieścić wyliczoną przez funkcję wartość, np. zapamiętać pod jakąś zmienną, wydrukować itp. Własność ta dotyczy zarówno funkcji standardowych Basica (np. SIN, ABS), jak i funkcji definiowanych przez programistę.

A oto przykład definicji i wywołania funkcji bezparametrowej:

```
 5 LET x=0
10 DEF FN a()=x+1
20 PRINT FN a()
```

W wielu przykładach umieszczonych w tym rozdziale użyto funkcji RND, która w wyniku daje losowo wybraną liczbę z zakresu od 0 do 1 (nigdy nie daje wartości równej dokładnie 1, niekiedy jednak daje 0). W komputerze trudno zorganizować „naprawdę" losowe wybieranie liczb. Funkcja RND daje nie całkiem losowe liczby, bowiem powtarza ona sekwencję 65536 liczb.

Z funkcją RND współpracuje instrukcja RANDOMIZE. Funkcja RND daje kolejne liczby z ciągu 65536 wartości, natomiast funkcja RANDOMIZE ustawia wskaźnik, według którego funkcja RND pobiera kolejne liczby z ciągu, na ustalony element sekwencji, np.:

```
  10 RANDOMIZE 100: REM pobieran
e beda liczby poczawszy od setne
go elementu
  20 PRINT RND,RND
  30 GO TO 10
```

Instrukcja RANDOMIZE lub RANDOMIZE 0 ustawia wskaźnik na wartość obliczoną na podstawie czasu przez jaki komputer był włączony*). Wykonanie sekwencji:

```
10 RANDOMIZE
20 PRINT RND
```

daje więc wartość wybraną losowo, bowiem nie można z góry określić w jakim momencie czasu instrukcje te zostaną wykonane. Jeśli jednak wartość losowa jest wyznaczona w programie w pętli, a czas wykonania pętli jest stały, to sekwencja taka daje zawsze podobne wartości np.:

```
10 RANDOMIZE
20 PRINT RND
30 GO TO 10
```

*) Komputer mierzy ten czas i wynik zapisuje w komórkach pamięci (FRAMES) o adresach: 23672, 23673, 23674. Zawartość komórki 23672 zwiększa się o jeden co 0,02 s, komórki 23673 co 5,12 s, a komórki 23674 co około 21 min).

Można ten problem częściowo rozwiązać, np. tak:

```
 10 IF (RND>0.5) AND (PEEK (236
72)<100) THEN RANDOMIZE
 20 PRINT RND
 30 GO TO 10
```

Można także wykorzystać komórki FRAMES do zrealizowania własnego generatora liczb losowych. Opracowanie lepszych generatorów losowych pozostawia się Czytelnikowi [32].

Przetwarzanie tekstów 2.2.8

Tekst w języku Basic ZX Spectrum to ciąg (łańcuch) dowolnych znaków ujętych w cudzysłowy, np.: "ala?". Tekstem jest więc także pojedynczy znak, np. "A" albo " ". Wreszcie tekst może być pusty, np.: "" (między cudzysłowami nie ma znaku odstępu — spacji).

Interpreter zapisuje w pamięci każdy znak w pojedynczym bajcie jako liczbę. Liczba ta jest nazywana k o d e m z n a k u. W ZX Spectrum stosuje się przyjęte na całym świecie liczbowe wartości znaków, tzw. kod ASCII (patrz dodatek A). Basic ma standardowe funkcje, które zmieniają znak na odpowiadającą mu liczbę (funkcja CODE), oraz liczbę na znak (funkcja CHR$), np. znak „a" ma kod 97, a więc instrukcja

PRINT CODE "a"

wydrukuje liczbę 97. Parametrem funkcji CODE może być cały ciąg znaków, ale i tak funkcja ta wykona się tylko dla pierwszego znaku z całego ciągu, np.:

PRINT CODE "ala"

wydrukuje liczbę 97. Natomiast instrukcja:

PRINT CHR$ 97

wydrukuje literę a (literę o kodzie 97).

Argumentem funkcji CHR$ mogą być także tzw. z n a k i s t e r u j ą c e np.: TAB, FLASH, INK itd (patrz p. 4.1). Dlatego np. wykonanie instrukcji:

PRINT CHR$ 18+CHR$ 1; "TEKST"

jest równoważne instrukcji:

PRINT FLASH 1; "TEKST"

ponieważ kod 18 odpowiada opcji FLASH.

Jeżeli Czytelnik sięgnie do dodatku A zauważy zapewne, że tylko niektórym liczbom (kodom) odpowiadają znaki, które mogą zostać wydrukowane, tzn. małe i duże litery, cyfry i znaki takie jak dwukropek, nawias, itd. Znaki zapisywane

są w pojedynczym bajcie, można zatem zapisać 256 różnych znaków. Znaków widocznych jest 94. Pozostałe ,,wolne" kody zostały przypisane znakom sterującym wyświetlaniem na ekranie, drukowaniem na drukarce oraz słowom kluczowym języka Basic.

Z dotychczasowej lektury tego rozdziału wynika, że:

— tekst może być zapamiętany w tzw. zmiennej tekstowej (nazwa takiej zmiennej musi składać się z pojedynczej litery i znaku $),

— tekst może być wydrukowany instrukcją PRINT: ,,bezpośrednio", np. PRINT "ala ma kota"; można też wydrukować tekst zapamiętany pod zmienną tekstową np.:

```
10 LET a$ = "ala ma kota"
20 PRINT a$
```

— tekst może być wczytany z klawiatury instrukcją INPUT, np.:

```
10 INPUT a$
```

lub

```
10 INPUT LINE a$
```

lub ze zbioru DATA instrukcją READ, np.:

```
15 READ a$
```

Język Basic ZX Spectrum pozwala na wykonywanie na tekstach operacji. Z tekstu można ,,wyciąć" jego fragment:

```
"abcdefgh" (1 TO 5)
 ↑          ↑    ↑
 |          od którego, do którego znaku będzie wycinany
tekst, który ma być przetwarzany
```

Na przykład: "abcdefgh" (1 TO 5) jest równe "abcde", natomiast "abcdefgh" (5 TO 7) — daje w wyniku "efg". Jeżeli pominie się pierwszy argument (w tym przypadku 5), to interpreter przyjmie 1. Nie podanie drugiego argumentu jest równoważne podaniu argumentu równego całkowitej długości tekstu, np.:

— zamiast "abcd" (1 TO 3) można napisać "abcd" (TO 3) — wynik "abc"

— zamiast "abcd" (2 TO 4) można napisać "abcd" (2 TO) — wynik "bcd"

— zamiast "abcd" (1 TO 4) można napisać "abcd" (TO) — wynik "abcd"

Oto kolejne przykłady użycia mechanizmu ,,wycinania" części tekstu:

"abcd" (3 TO 7) — interpreter zasygnalizuje błąd (3 SUBSCRIPT WRONG!) bowiem ciąg składa się tylko z 4 znaków, a nie z co najmniej 7.

"abcd" (8 TO 7) — daje w wyniku tekst pusty"" (nie ma błędu),

"abcd" (1 TO 0) — daje tekst pusty "",

"abcd" (3 TO 3) — daje pojedynczy znak, w tym przypadku "c".

Operację wycinania można stosować do tekstów napisanych explicité, tzw. stałych tekstowych (tak jak to zrobiono wyżej) oraz do zmiennych tekstowych, np.

```
10 LET a$="xxxx"
20 LET b$=a$(2 TO 3)
30 PRINT a$
40 PRINT b$
```

W Basicu ZX Spectrum można także zmienić ,,część" zmiennej tekstowej, np.

```
10 LET a$="Ala ma kota"
20 LET a$(8 TO 11)="psa "
30 PRINT a$
```

W wyniku wykonania tego programu otrzyma się wydruk: "Ala ma psa", bowiem linia 20 oznacza: w zmiennej tekstowej a$ w miejsce znaków od 8 do 11 podstaw znaki pobrane z ciągu znaków (lub wyrażenia, które daje w wyniku ciąg znaków) znajdującego się po znaku równości (po prawej stronie).

Teksty można także konkatenować (składać) ze sobą, np.:

```
10 LET a$="ala"
20 LET a$=a$+"bama"
30 PRINT a$
```

Na ekranie otrzymamy tekst "alabama".

Oto inny przykład:

```
10 LET a$="ktora "
20 LET b$="godzina?"
30 PRINT a$+b$
```

Komputer wydrukuje: "która godzina?".

Teksty mogą być także porównywane ze sobą. Tekst, który znajdzie się wcześniej w uporządkowaniu alfabetycznym oznacza się znakiem <, np.: warunek "alabama" < "ela" jest prawdziwy, również prawdziwy jest: "ala" < = "ela" natomiast fałszywy jest warunek "ala" > "ela" oraz "ala" = "ela".

Interpreter porównuje znaki porównując ich kody. Dlatego ciąg znaków zaczynających się od dużej litery, np.: Z, będzie uznany za "mniejszy" niż zaczynający się od małej litery, np. "a".

Podany niżej fragment programu stanowi przykład zastosowania opisywanych możliwości ZX Spectrum:

```
 100 PRINT AT 1,0;"podaj odpowie
dz: tak/nie?"
 110 INPUT a$
 120 IF a$="tak" THEN LET a=1: G
O TO 150
 130 IF a$<>"nie" THEN GO TO 100
 140 LET a=0
 150 IF a=1 THEN PRINT "wpisano
TAK": STOP
 160 IF a=0 THEN PRINT "wpisano
NIE": STOP
```

Program wczytuje odpowiedź, zapamiętuje ją (zmienna A) oraz nie daje się zbyć byle jaką odpowiedzią.

Basic ZX Spectrum ma także trzy bardzo przydatne funkcje: LEN, STR$ i VAL. Funkcja LEN pozwala ,,zmierzyć" ile znaków zawiera zmienna tekstowa. Na przykład po wpisaniu fragmentu programu:

```
10 LET a$="ala"
20 PRINT LEN a$
```

na ekranie wyświetli się cyfra 3 (tekst "ala" zawiera trzy litery). Jeżeli w linii 10 za zmienną a$ podstawiono by np.: tekst "elżbieta", funkcja LEN miałaby wartość 8.

Funkcja STR$ zamienia liczbę na ciąg znaków, np.:

```
10 LET a$ = STR$ 125:PRINT a$
```

lub

```
10 LET b = 105:LET a$ = STR$ b:PRINT a$
```

Liczba jest zapisywana w pamięci w pięciu bajtach w pewien określony sposób. W czasie wykonywania funkcji STR$ interpreter zamienia liczbę na odpowiedni ciąg kodów znaków, który może być zapamiętany w zmiennej znakowej lub wydrukowany.

Funkcja VAL jest przeciwna do STR$. Zamienia ciąg znaków na liczbę, np.:

```
10 LET a=VAL "123.4": PRINT a
20 LET a$="1e5"
30 LET a=VAL a$: PRINT a
```

Parametrem funkcji VAL może być ciąg znaków (liczba lub dowolne wyrażenie arytmetyczne). Najlepiej możliwości funkcji VAL zilustrują podane niżej przykłady:

— VAL ("3*2") daje w wyniku liczbę 6

— VAL ("4*5"+"+2") daje w wyniku liczbę 22, najpierw bowiem wykona się operacja + (łączenia dwu tekstów w jeden), która da w wyniku tekst "4*5+2" następnie zostanie obliczone wyrażenie i wynik zamieniony (przekodowany) w liczbę.

Dotychczas pokazano jeden sposób, w jaki tekst może być zapisany w pamięci, a mianowicie podstawiony do zmiennej znakowej. Znaki mogą być także umieszczone w tablicy znaków. Tablicę taką deklaruje się instrukcją DIM, przy czym nazwa tablicy znakowej składa się z pojedynczej litery i znaku $. Elementem takiej tablicy jest pojedynczy znak. Tablice znakowe zorganizowane są podobnie jak tablice liczbowe, z tą tylko różnicą, że każdy element z tablicy liczbowej zajmuje 5 bajtów, a tablicy znakowej — 1 bajt.

Jeżeli np. zadeklaruje się tablicę a$ (3, 5), to tablica ta ma 3 wiersze, z których każdy zawiera 5 znaków. Można się odwołać do dowolnego znaku tablicy w następujący sposób:

```
10 DIM a$(3,5)
20 LET a$(2,1)="a"
30 PRINT a$(2,1)
```

W linii 20 znak "a" jest umieszczony jako pierwszy znak w drugim wierszu tablicy a$. W linii 30 jest on drukowany. Można także odwołać się do całego wiersza tablicy, np.:

```
10 DIM a$(3,5)
20 INPUT a$(2)
30 IF a$(2)="abcde" THEN LET a
$(3)=a$(2)
40 PRINT a$(3)
50 PRINT a$(2)
```

W powyższym programie w liniach 20, 30 i 50 znajdują się odwołania do całego wiersza (pięciu znaków) tablicy.

Z tablicy można także wycinać fragmenty tekstu, podobnie jak dzieje się to ze zmiennymi tekstowymi, np.:

```
10 DIM a$(2,5)
20 LET a$(1)="abcd"
30 PRINT a$(1)
40 PRINT a$(1,2 TO 4)
41 REM
50 PRINT a$(1)(2 TO 4)
```

Wyrażenie a$ (1,2 TO 4) z linii 40 oznacza, że z pierwszego wiersza należy wyciąć znaki od drugiego do czwartego. Instrukcję z linii 40 można zapisać także w następujący sposób (obie formy dla interpretera są identyczne):

```
40 PRINT a$(1) (2 TO 4)
```

Język Basic ZX Spectrum zawiera instrukcje typowego języka Basic, np. LET, PRINT, FOR, GO TO itp. Ma także instrukcje specyficzne tylko dla ZX Spectrum Są to głównie instrukcje graficzne, np.: PLOT, DRAW itd. (patrz rozdz. 3), dodatkowe instrukcje, które mogą być zrealizowane ze względu na konfigurację komputera, np. BEEP, OUT, CLS itd., a także inne instrukcje będące rozszerzeniem języka, np. PAUSE, USR, LOAD i inne.

* Jak wprowadzać program? Praca z edytorem ekranowym 2.3

Komputer po włączeniu zasilania wyświetla komunikat „© Sinclair research Ltd. 1982", a następnie oczekuje na instrukcje użytkownika. Informuje o tym migający w lewym dolnym rogu ekranu ciemny prostokąt z literą, tzw. kursor.

Można wtedy wpisywać za pomocą klawiatury linie programu lub instrukcje do natychmiastowego wykonania. Każdą wprowadzoną linię kończy się wciskając klawisz ENTER. Treść wprowadzanej linii pojawia się w czasie wprowadzania w dolnej części ekranu sukcesywnie, kursor zaś przesuwa się w prawo. Jeśli linia programu nie mieści się w jednym wierszu ekranu, to zostaje ona przesunięta w górę (podobnie jak dzieje się to przy zmianie wiersza w maszynie do pisania), a tekst jest wprowadzany dalej. Po wciśnięciu klawisza ENTER interpreter sprawdza, czy wprowadzana linia jest poprawna, a następnie wprowadza ją do pamięci programu (linia ta jest drukowana w górnej części ekranu) lub wykonuje, jeżeli linia nie ma numeru.

Gdy linia jest błędna, to w miejscu wykrycia pierwszego błędu pojawia się dodatkowy migający kursor (linia nie jest kopiowana do górnej części ekranu, ani wykonywana). Możemy następnie przesunąć kursor w dowolne miejsce linii programu za pomocą klawiszy ⇨ (przesuwanie w prawo) i ⇦ (przesuwanie w lewo). Możemy usunąć dany znak, cyfrę lub słowo kluczowe (SYMBOL SHIFT oraz 0 wciśnięte równocześnie lub DELETE w ZX Spectrum +) oraz wprowadzić nowe znaki itd. Gdy uważamy, że linia programu jest gotowa, to wciskamy klawisz ENTER.

Przy wprowadzaniu znajdują zastosowanie następujące reguły:
— linia jest wprowadzana do programu, który jest porządkowany, wg rosnących numerów linii (jeśli wprowadzimy najpierw linię 20, a następnie 10, w programie linia 10 będzie wcześniej),
— numer linii musi być liczbą naturalną z zakresu od 1 do 9999,
— jeśli linia o danym numerze istnieje już w programie, to jest ona zastępowana nowo wprowadzoną linią,
— jeśli napiszemy jedynie numer linii oraz ENTER, to linia o podanym numerze jeżeli była w programie, zostanie usunięta.

Chcąc poprawić linię błędnie wprowadzoną do programu możemy powtórnie, poprawnie napisać linię o tym samym co błędna numerze. Możemy także skorzystać z możliwości którą daje wciśnięcie klawisza EDIT (CAPS SHIFT 1).

Linie programu wprowadzone już do komputera są wyświetlane w górnych wierszach ekranu. Można zauważyć, że przy jednej z nich (tej wprowadzonej ostatnio) jest wyświetlany, zaraz za numerem linii, znaczek „[>]". Jest to tzw. **kursor programu**. Linia, na którą wskazuje, zostanie przepisana w dolną część ekranu, po naciśnięciu klawisza EDIT. Można tak skopiowaną linię poprawić w analogiczny sposób, jak przy wprowadzaniu nowej linii. Wciśnięcie klawisza ENTER powoduje zapamiętanie poprawionej linii (można nacisnąć EDIT oraz ENTER, gdy stwierdzi się, że nie trzeba nic zmieniać).

Kursorem programu można przesunąć na następną linię za pomocą znaku ⇩ (CAPS SHIFT i 6) lub na poprzednią linię — ⇧ (CAPS SHIFT i 7). Jeśli usunięto linię, na którą pokazywał kursor programu „[>]", to znika on z ekranu, ale wciśnięcie lub powoduje wyświetlenie go ponownie przy linii: odpowiednio poprzedniej lub następnej w stosunku do usuniętej.

Program znajdujący się w pamięci może być wyświetlany na ekranie po wykonaniu w trybie natychmiastowym instrukcji LIST. Jeśli program nie mieści się na ekranie, to w ostatniej linii ekranu pojawia się pytanie: Scroll? Jeśli odpowiedzią będzie naciśnięcie klawisza BREAK, N, STOP lub SPACE dalsze wyświetlanie jest przerywane. Dowolny inny klawisz powoduje wyświetlanie następnej porcji programu.

Możliwe jest wyświetlanie programu począwszy od linii o określonym numerze — dyrektywa LIST *n*, gdzie *n* — nr linii, np. LIST 150. Nie można natomiast wyświetlić programu „od linii do linii" (możliwe w wielu komputerach, ale nie w ZX Spectrum). Treść programu można wydrukować na drukarce za pomocą instrukcji LLIST lub LLIST *nr linii*.

* Uruchomienie programu w języku Basic: RUN, NEW, BREAK, CLEAR, CONTINUE 2.4

Jak uruchomić program? 2.4.1

Jak? Wystarczy wpisać instrukcję RUN (klawisz R) oraz ENTER i program zaczyna działać. Interpreter pobiera wtedy kolejne linie programu od pierwszej aż do ostatniej lub do momentu wykonania instrukcji STOP. Analizuje i wykonuje zaprogramowane operacje.

Uruchomienie programu od wybranej linii realizuje instrukcja

```
RUN n
     ↑
     numer linii, od której wykonanie programu się rozpocznie
     (może być również wyrażenie)
```

Jeżeli w czasie wykonywania programu interpreter wykryje błąd (np. próbę dzielenia przez zero), to wykonywanie programu przerywa się i w dwu dolnych liniach ekranu (linie te nazywane są niekiedy oknem systemowym) interpreter drukuje komunikat o błędzie (wszystkie komunikaty systemu podano w dodatku D). W linii tej jest drukowany także komunikat o poprawnym zakończeniu programu, oczywiście poprawnym z punktu widzenia interpretera. Znaczy to, że wszystkie instrukcje dały się interpretować. Nie oznacza to, że program działa tak, jak tego sobie życzy programista. Komunikat ten ma następującą postać:

```
0 OK,  150:1
↑ ↑     ↑ ↑
| |     | numer instrukcji w linii
| |     numer ostatniej wykonanej linii
| treść komunikatu
numer komunikatu
```

Gdy wykonanie programu kończy się instrukcją STOP, komunikat końcowy ma postać:

9 STOP statement, 215:1

numer instrukcji w linii

numer ostatniej wykonanej linii programu

Interpreter pamięta numer linii programu, w której wystąpiła instrukcja STOP. Program zatrzymany na skutek wykonania takiej właśnie instrukcji może być interpretowany dalej. W tym celu należy napisać instrukcję:

CONTINUE (ENTER)

Opisane wyżej własności można sprawdzić za pomocą prostego programu:

```
10 PRINT "linia nr 10 ": STOP
20 PRINT "linia nr 20 ": STOP
30 PRINT "linia nr 30 ": STOP
40 PRINT "koniec"
```

Po każdym zatrzymaniu się programu należy w celu kontynuowania wykonania wcisnąć klawisz CONT, aż do zakończenia programu (komunikat OK).

Opisana tu możliwość wykonywania programu „krokowo" jest bardzo przydatna w czasie testowania i uruchamiania nowych programów.

Program można także uruchomić instrukcją GO TO (a podprogram instrukcją GO SUB). Napisanie w trybie natychmiastowym, tzn. bez numeru linii, instrukcji: GO TO *nr linii* (lub GO SUB) spowoduje, że program zaczyna się wykonywać od podanej linii. GO TO 0 oznacza, że wykonywanie zacznie się od pierwszej linii programu, niezależnie od jej numeru.

Instrukcja GO TO pozwala także na wykonywanie programu „od środka". Dociekliwy Czytelnik zapyta czy instrukcja RUN jest równoważna GO TO? Otóż nie! Instrukcja RUN oprócz tego, że powoduje wykonanie programu, realizuje dodatkowe funkcje: dokonuje „przewinięcia" zbioru DATA (analogicznie jak RESTORE) oraz kasuje obszar zmiennych programu (obszar ten jest opisany w dalszej części) i ustawia wartości niektórych zmiennych systemowych (tzn. komórek pamięci używanych przez programy umieszczone w pamięci ROM).

Jak przerwać wykonanie programu 2.4.2

W ZX Spectrum można zatrzymać wykonywanie programu napisanego w języku Basic za pomocą klawisza BREAK. Należy wcisnąć jednocześnie CAPS SHIFT oraz BREAK i niekiedy chwilę przytrzymać (np. 1 s, a w przypadku, gdy aktualnie wykonuje się instrukcja BEEP lub DRAW, nawet dłużej). Wykonanie programu jest wtedy przerywane. Drukuje się komunikat:

D BREAK — CONT repeats, *nr*:*k*

numer instrukcji w tej linii

numer ostatnio wykonanej linii

lub

L BREAK into program, *nr*:*k*

Wykonanie programu można wznowić instrukcją CONTINUE (analogicznie jak w przypadku instrukcji STOP). Gdy program został przerwany w czasie wykonywania operacji we/wy (komunikat BREAK — CONTINUE repeats), to po naciśnięciu klawisza CONTINUE przerwana instrukcja jest powtarzana od początku. W innym przypadku (komunikat BREAK into program) po wciśnięciu CONTINUE interpreter zaczyna wykonanie od następnej instrukcji po instrukcji przerwanej.

Jeżeli chcemy przerwać program, w którym wykonuje się właśnie instrukcja INPUT, to można to uczynić naciskając SYMBOL SHIFT oraz STOP lub CAPS SHIFT i 6 zamiast informacji, np. liczby, itd.

Jeżeli interpreter oczekuje na tekst, tzn. wykonuje się np. instrukcja:

10 INPUT a$

to w najniższej linii ekranu wyświetla się migający kursor ujęty w cudzysłowy. Muszą one być skasowane (za pomocą klawisza ⇦ oraz DELETE). Następnie można wcisnąć STOP.

Jeżeli cudzysłowów nie ma, tzn. wykonuje się np.:

10 INPUT A

lub

10 INPUT LINE a$

to można nacisnąć w pierwszym przypadku bezpośrednio STOP, a w drugim CAPS SHIFT 6.

Klawisz BREAK może być zablokowany. Blokadę działania klawisza BREAK stosuje się często w programach (np. grach), gdy producent chce, aby jego program był nieprzerywalny. Klawisz BREAK zostanie zablokowany, jeżeli zawartość pary komórek pamięci o adresie 23613, 23614 (zmienna systemowa o nazwie ERR SP) zostanie zmniejszona o dwa. Zmienna ta zawiera adres programu napisanego w kodzie maszynowym i umieszczonego w pamięci ROM, który wykonuje się, gdy nastąpi błąd w programie w Basicu.

Uwaga! *Jeżeli klawisz BREAK zostanie w ten sposób zablokowany, to każdy błąd wykonania programu spowodować może zawieszenie się interpretera — trzeba będzie wtedy odłączyć zasilanie lub wcisnąć klawisz RESET, by uruchomić komputer na nowo. Pociąga to za sobą utratę wszystkich informacji zapamiętanych w pamięci RAM*

Każdorazowe wciśnięcie klawisza RUN powoduje odtworzenie standardowej zawartości zmiennej ERR SP. Dlatego instrukcja:

POKE 23613, PEEK 23613-2

blokująca klawisz BREAK powinna być umieszczona jako pierwsza instrukcja programu.

Kasowanie programu napisanego w języku Basic 2.4.3

Niepotrzebny program może być usunięty z pamięci za pomocą instrukcji NEW. Program usuwany jest także na skutek wykonania instrukcji LOAD. W tym przypadku stary program jest zastępowany nowym.

W ZX Spectrum pamięć RAM zajmuje 48 kbajtów, od adresu 16384 do adresu 65535 (największy możliwy adres w Spectrum)*). Pamięć RAM, która może być wykorzystana przez interpreter Basica jest ograniczona. Górna granica (największy adres) pamięci jest zapamiętana w zmiennej systemowej RAMTOP (dwa bajty pamięci o adresach 23730, 23731). Po włączeniu zasilania wartość zmiennej RAMTOP jest ustawiana na wartość 65367. Można ją odczytać pisząc

PRINT PEEK (23730)+PEEK (23731)*256

Wartość zmiennej RAMTOP można zmienić za pomocą instrukcji CLEAR *liczba*. Instrukcja CLEAR ustawia wartości początkowe zmiennych używanych przez interpreter Basica oraz kasuje obszar zmiennych programu w Basicu — patrz 2.5.1. Instrukcja w postaci CLEAR *liczba* ustawia także nową wartość zmiennej systemowej RAMTOP. Na przykład, po wykonaniu

CLEAR 50000

zmienna RAMTOP przyjmuje wartość 50000. Instrukcja CLEAR nie usuwa z pamięci komputera programu napisanego w Basicu.

Uruchamianie i testowanie programu 2.4.4

Uruchamianie i testowanie programu jest łatwe, interpreter bowiem dostarcza użytkownikowi mechanizmy, które znacznie ułatwiają proces uruchamiania i testowania programu.

Program w języku Basic, zapisany w pamięci RAM, składa się z dwu części: obszaru programu oraz obszaru zmiennych. Po wprowadzeniu programu do pamięci (z klawiatury, a niekiedy z magnetofonu), obszar zmiennych jest pusty.

*) W wersji 48 k.

Obszar ten jest wypełniany w czasie wykonywania programu. W obszarze zmiennych są umieszczane wszystkie zmienne i tablice, które znajdują się w kolejno interpretowanych liniach programu. Obszar zmiennych pozostaje w pamięci po przerwaniu wykonania programu lub jego zakończeniu, np. jeżeli wykona się program:

```
10 LET A = 5
20 LET B = SIN A
```

to po jego zakończeniu możemy w trybie natychmiastowym wydrukować zmienne A i B, zmienne te bowiem znajdują się w pamięci w obszarze zmiennych, np.:

```
PRINT A, B
```

Można także wykonać na tych zmiennych operacje, np.:

```
LET C = A+B
```

Zmienna C znajdzie się wtedy w obszarze zmiennych, choć nie było jej w programie.

Można także wcześniej ustawić w trybie natychmiastowym wartości zmiennych, a następnie uruchomić instrukcją GO TO fragment programu, który z nich korzysta. Wówczas w trybie natychmiastowym piszemy:

```
LET A = 5 : LET B = 3
GO TO 20
```

a program może mieć postać:

```
10 LET A = 2:LET B = 1
20 LET C = A+B
30 PRINT C
```

Program został uruchomiony instrukcją GO TO, a nie instrukcją RUN. Instrukcja RUN kasuje bowiem najpierw obszar zmiennych.

Reasumując: Interpreter pozwala nam na uruchomienie programu krokowo. Wygodne jest na czas uruchomienia dopisać do programu instrukcję STOP (jedną lub kilka), dzięki czemu program będzie się zatrzymywał w miejscach, w których ją umieścimy. Programista będzie mógł wówczas sprawdzić, czy wartości zmiennych są poprawne, czy fragment programu wykonuje się dobrze dla określonych wartości zmiennych, itd. Gdy program będzie poprawny, to niepotrzebne już instrukcje STOP można usunąć.

Zamiast instrukcji STOP można użyć instrukcji PAUSE. Instrukcja ta powoduje zawieszenie wykonywania programu na pewien czas, np.:

```
PAUSE 100
        ↑
czas, w którym program będzie zawieszony
```

Parametr instrukcji PAUSE razy 1/50 daje czas w sekundach, a więc w tym przypadku wykonanie zostanie zawieszone na około 100x1/50 = 2 s. Instrukcja PAUSE 0 oznacza zawieszenie programu na czas nieokreślony. Jednakże przy-

ciśnięcie dowolnego klawisza powoduje przerwanie każdej instrukcji PAUSE i wznowienie wykonywania programu.

Interpreter pozwala także testować fragment programu. Jeżeli program został podzielony na podprogramy, to wygodnie jest uruchamiać podprogramy osobno. Jest to kolejna korzyść pisania podprogramów. Można zamiast instrukcji GO TO *nr linii* uruchomić podprogram wykonując w trybie natychmiastowym instrukcję GO SUB *nr linii*. Interpreter wykona wtedy instrukcje podprogramu począwszy od podanej linii, a po wykonaniu instrukcji RETURN, kończącej podprogram, wykonanie zakończy się z komunikatem OK.

Wskazane jest także by sprawdzać, czy program odpowiednio reaguje na niepoprawne dane (np. wprowadzane z klawiatury). Program należy testować dla granicznych wartości danych. Na przykład, jeżeli program ma się wykonywać dla danych z zakresu od 0 do 100, to należy sprawdzić czy wykona się poprawnie dla 0 i dla 100 oraz dla jednej lub kilku wartości ze środka pętli.

* * Jak jest zapamiętany program napisany w Basicu? 2.5

Zmienne systemowe PROG, VARS, ELINE 2.5.1

Adres początku programu napisanego w języku Basic i znajdującego się w pamięci jest umieszczony w komórkach pamięci o adresie 23635 i 23636 (adres zajmuje 16 bitów — stąd 2 bajty). Komórki te są nazywane zmienną systemową PROG. Adres początku programu można obliczyć:

PRINT "Początek programu", PEEK (23635)+256*PEEK(23636)

Miejsce, w którym zaczyna się program może się zmienić. Zależy ono od tego, czy używa się microdrive'ów oraz tzw. strumieni danych (patrz rozdz. 4). Dlatego adres początku programu nie jest ustalony, ale zapamiętywany każdorazowo w zmiennej systemowej PROG.

Miejsce, w którym kończy się program w Basicu można odczytać na podstawie zawartości zmiennej systemowej VARS — z komórki o adresach 23627, 23628. Zmienna VARS wskazuje na obszar, w którym zaczynają się zmienne programu. Obszar zmiennych jest umieszczony za obszarem programu.

Adres początku zmiennych można obliczyć:

PRINT "Początek zmiennych", PEEK (23627)+256*PEEK(23628)

Z kolei za obszarem zmiennych jest umieszczony bufor edytora ekranowego, czyli pamięć używana przez edytor ekranowy w czasie wprowadzania tekstu programu (lub instrukcji do natychmiastowego wykonania). Adres początku bufora zapamiętany jest w zmiennej ELINE (bajty o adresach 23641, 23642).

Na podstawie informacji umieszczonych w zmiennych PROG, VARS i ELINE można obliczyć ile pamięci zajmuje program oraz zmienne.

```
1000 DEF FN a(n)=PEEK n+256*PEEK
 (n+1)
1010 PRINT "Program zajmuje ";FN
 a(23627)-FN a(23635)-1
1020 PRINT "Zmienne zajmuja ";FN
 a(23641)-FN a(23627)-1
```

Należy pamiętać, że po wprowadzeniu programu do pamięci obszar zmiennych jest zazwyczaj pusty. Obszar ten wypełnia się w czasie wykonywania programu i pozostaje w pamięci po jego zakończeniu (kasowany jest po wykonaniu RUN, CLEAR, NEW — łącznie z programem).

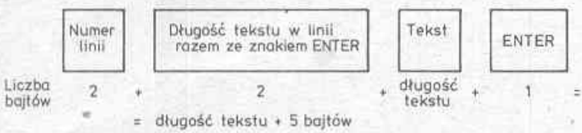
Podany wyżej fragment programu należy dopisać do programu, który chcemy ,,zmierzyć", tak by wykonał się na końcu. Jeśli dopisze się linię:

1030 PRINT "Wolne"; FN A(23613)—FN A(23653)

otrzyma się liczbę bajtów wolnej pamięci.

Jak jest zorganizowany obszar programu? 2.5.2

,,Obraz" programu zapamiętanego w pamięci jest w zasadzie podobny do tego, co można zobaczyć na ekranie po wykonaniu instrukcji LIST, choć są pewne różnice. Każda linia jest zapamiętana w pamięci w sposób pokazany na rys. 9.



Rys. 9. Sposób zapamiętania linii programu

Pole ,,numer linii" zajmuje 2 bajty pamięci. Na podstawie ich zawartości możemy obliczyć numer linii, w tym celu zdefiniowano funkcję M.

```
1010 DEF FN m(a)=256*PEEK (a)+PE
EK (a+1): REM a-adres poczat
ku linii w pamieci
1020 LET Adres=PEEK (23635)+256*
PEEK (23636)
1030 PRINT FN m(Adres)
```

Powyższy program wydrukuje numer pierwszej linii w programie. W linii 1020 wyznacza się adres programu korzystając ze zmiennej systemowej PROG. Dwa kolejne bajty umieszczone w pamięci począwszy od obliczonego adresu zawierają numer pierwszej linii programu.

Uwaga! *W polu „numer linii" bardziej znaczący jest pierwszy bajt, mniej znaczący — drugi. Odwrotnie niż ma to miejsce zazwyczaj w ZX Spectrum, np.: w zmiennych systemowych lub polu „długość tekstu".*

Jeżeli obliczony jest adres początku linii w pamięci (niech zmienna o nazwie Adres zawiera obliczony adres), to długość linii można wyznaczyć następująco:

PRINT PEEK (Adres+2)+256*PEEK(Adres+3)+4

↑ adres drugiego, bardziej znaczącego bajtu

↑ adres pierwszego bajtu pola „długości linii"; bajt mniej znaczący

Adres początku następnej linii można obliczyć:

LET Adres = Adres+PEEK(Adres+2)+256*PEEK(Adres+3)+4

adres następnej linii = adres poprzedniej + długość linii

Niżej podano przykładowy program służący do automatycznego przenumerowywania linii programu w Basicu, począwszy od numeru 10 i zwiększając numery co 10. Należy go dopisać na końcu przenumerowywanego programu, np. za pomocą instrukcji MERGE, oraz uruchomić instrukcję GO TO 9000 lub RUN 9000.

```
9000 DEF FN h(n)=INT (n/256): RE
M Funkcja wyznacza bardziej  zn
aczacy bajt liczby n
9010 DEF FN l(n)=n-256*FN h(n):
REM Funkcja wyznacza mniej znacz
acy bajt liczby n
9020 DEF FN k(n)=n+4+PEEK (n+2)+
256*PEEK (n+3): REM Funkcja wyzn
acza adres kolejnej linii. n-adr
es linii biezacej
9030 DEF FN b()=PEEK (23635)+256
*PEEK (23636): REM Funkcja wyzna
cza adres pierwszej linii progra
mu
9050 DEF FN c(n)=256*PEEK (n)+PE
EK (n+1): REM Funkcja wyznacza n
umer linii zapamietanej od adres
u n
9100 LET Adres=FN b()
9110 LET koniec=9000
9120 LET Numer=10
9130 PRINT "Stary nr=";FN c(Adre
s);" Nowy nr=";Numer
9140 POKE Adres,FN h(Numer): POK
E (Adres+1),FN l(Numer): REM Wpi
sanie nowego numeru
9150 LET numer=Numer+10
9155 LET Adres=FN k(Adres): REM
Adres nastepnej linii
9160 IF FN c(Adres)<Koniec THEN
GO TO 9130
```

Podany program, aczkolwiek użyteczny, nie uwzględnia faktu, że po przenumerowaniu nie będą się zgadzały numery linii w instrukcjach GO SUB, GO TO. Automatyczne poprawianie tych numerów wymagałoby analizowania i poprawiania treści linii programu, co znacznie skomplikowałoby program. Numery linii w przenumerowanym programie można poprawić „ręcznie", wykorzystując wydrukowane na ekranie stare i nowe numery linii.

A jak umieszczony jest w pamięci tekst linii programu?

Kolejne bajty tekstu linii są umieszczone tak, jak były wprowadzone z klawiatury. Znakom oraz słowom kluczowym odpowiadają ich kody (takie, jakiej zostały podane w dodatku B). Słowom kluczowym odpowiadają więc jednobajtowe kody, np. zamiast tekstu GO SUB w pamięci jest umieszczony bajt równy 237.

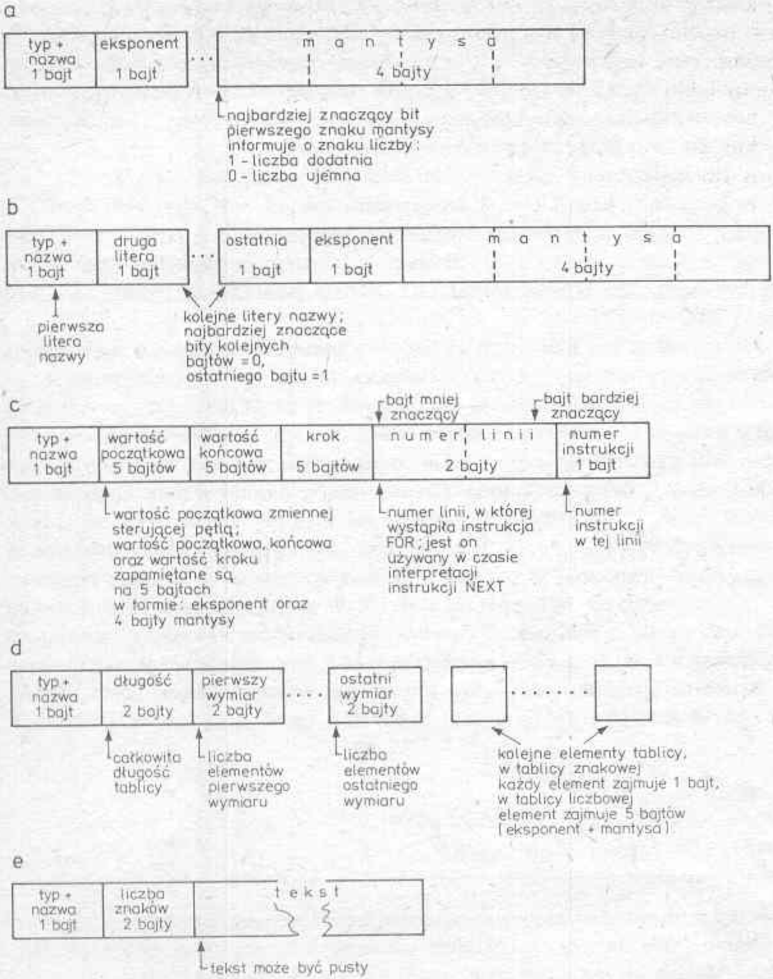
Od tej reguły jest jeden wyjątek. Jeżeli w tekście programu jest umieszczona tzw. stała liczbowa (czyli konkretna liczba), to jest ona zapamiętywana w ten sposób, że po znakach (cyfrach) składających się na tę liczbę, jest zapamiętany bajt o wartości 14 (informuje o końcu liczby). Następnie na pięciu bajtach zostaje zapamiętana wartość tej liczby w takim formacie, jaki jest używany w obliczeniach wykonywanych przez interpreter Basica, tzn. w formie zmiennoprzecinkowej (wyjątkiem są liczby całkowite z zakresu od 0 do 65335, które są zapisywane w postaci: 0, 0, dwa bajty, 0). Wartość liczby jest używana w czasie interpretacji programu. Dzięki temu, że jest ona obliczana w czasie wprowadzania programu do pamięci, program wykonuje się szybciej. W czasie wyświetlania (listowania) programu na ekranie używa się znakowej postaci liczby. Procedury umieszczone w pamięci ROM obsługujące dyrektywę LIST wyświetlają na ekranie kolejne znaki tekstu (zamiast kodów słów kluczowych wyświetlana jest pełna nazwa), np.: gdy w pamięci znajduje się kod równy 237, wyświetlany jest tekst GO SUB.

Jak jest zorganizowany obszar zmiennych? **2.5.3**

Zmienne w obszarze zmiennych są umieszczane sukcesywnie w czasie wykonywania programu. Gdy interpreter znajduje zmienną w interpretowanym programie, to przeszukuje obszar zmiennych. Jeżeli nie ma jeszcze takiej zmiennej, to jest ona w nim umieszczana.

W Basicu ZX Spectrum występują zmienne sześciu typów: zmienne proste (nazwy jedno- i wieloliterowe), tablice liczb, tablice znaków, zmienne tekstowe, zmienne indeksowe (występujące w instrukcji FOR). Nazwy zmiennych mogą składać się z pojedynczych liter. Wyjątek stanowią nazwy zmiennych prostych, składające się z wielu liter. Interpreter rozróżnia zmienne proste o nazwach jedno- i wieloliterowych jako dwa różne typy zmiennych.

Typ zmiennej jest zapisywany w pierwszym bajcie kilkubajtowego bloku pamięci przeznaczonego na zmienną w obszarze zmiennych. W pierwszym bajcie zapisuje się także nazwę zmiennej, jeśli nazwa ta jest jednoliterowa, lub pierwszy



Rys. 10. Sposób zapamiętania zmiennych języka Basic:

a — zmienna prosta — nazwa jednoliterowa, b — zmienna prosta — nazwa wieloliterowa, c — zmienna indeksowa, d — tablica (liczbowa lub znakowa), e — łańcuch znaków

Uwaga: tablica znakowa po zadeklarowaniu (instrukcja DIM) jest wypełniana znakami odstępu (spacjami), natomiast liczba jest zerowana. Uporządkowanie elementów jest następujące, np. dla tablicy zadeklarowanej DIM A (2,2) w pamięci znajdują się kolejno: A (1,1), A (1,2), A (2,1), A (2,2)

znak nazwy wieloliterowej. Typ zmiennej jest zapisywany w trzech bardziej znaczących bitach, nazwa w dalszych pięciu. W alfabecie ZX Spectrum znajduje się 26 liter, gdyż duże i małe litery umieszczone w nazwie są utożsamiane ze sobą. W pięciu bitach zapisuje się jedynie numer litery, np.: 1 dla A, 2 dla B, 26 dla Z:

— bity b7, b6, b5 — typ zmiennej,

— bity b4, b3, b2, b1, b0 — nazwa zmiennej.

Typy zmiennych:

dziesiętnie	binarnie	
2	010	zmienna tekstowa,
3	011	zmienna prosta (liczbowa) — nazwa jednoliterowa,
4	100	tablica liczb,
5	101	zmienna prosta — nazwa wieloliterowa,
6	110	tablica znaków,
7	111	zmienna indeksowa.

Jeżeli Adr zawiera adres pamięci, w której jest umieszczona zmienna, to kod typu zmiennej oraz nazwę (lub pierwsza litera nazwy) można wydrukować:

```
9000 LET typ=INT (PEEK (adr)/32)
9010 LET n$=CHR$ (PEEK (adr)-32*
typ+CODE ("A")-1)
9020 PRINT "typ-";typ
9030 PRINT "nazwa-";n$
```

Program ten drukuje tylko pierwsze litery nazwy jako duże litery.

Sposób w jaki umieszczone są w pamięci zmienne różnych typów podano na rys. 10.

* Grafika 3

* Wprowadzenie 3.1

Informacje zawarte w tym rozdziale pozwolą na zapoznanie się z możliwościami graficznymi ZX Spectrum. Zostaną tu omówione wszystkie instrukcje języka Basic tego mikrokomputera umożliwiające tworzenie własnych grafik oraz eleganckich wydruków. Rozdział ten jest ważny ponieważ:

— grafika jest bardzo istotnym elementem każdego programu; w przypadku gier i programów edukacyjnych jest często elementem najważniejszym,
— poznanie możliwości graficznych danego mikrokomputera pozwala na tworzenie na ekranie telewizora własnej grafiki,
— zrobienie własnej ciekawej szaty graficznej programu może dać programiście wiele satysfakcji.

* Wiadomości podstawowe 3.2

Obraz w ZX Spectrum jest generowany na podstawie zawartości pewnego obszaru pamięci RAM nazwanego pamięcią ekranu (ang. *display file*). Obszar ten rozpoczyna się od adresu 16384. Każdemu punktowi na ekranie (tzw. pixelowi) jest przyporządkowany jeden bit w pamięci obrazu. Obraz na ekranie składa się ze 192 wierszy po 256 punktów w każdym. Innymi słowy ekran jest macierzą 256 × 192 elementy. Na pamięć obrazu komputer potrzebuje więc 256 × 192 bity, tj. 6144 bajty (każdy bajt ma osiem bitów).

Mikrokomputer może sterować zapalaniem lub wygaszaniem na ekranie dowolnego punktu. Każdy punkt na ekranie może mieć kolor tła, jeśli jest wygaszony, lub kolor atramentu, jeśli jest zapalony. Jeżeli bit przyporządkowany do danego punktu ma wartość zero, to punkt ten jest wygaszony, a jeżeli ma wartość jeden, to punkt ten jest zapalony. W ten sposób jest zapamiętywany dowolny obraz.

ZX Spectrum ma możliwość generowania obrazów kolorowych. Dzieje się to tak, że na obraz zapamiętany w pamięci obrazu są nakładane a t r y b u t y. Atrybuty określają kolor w jakim są pisane znaki, czyli: kolor atramentu — INK, kolor tła — PAPER, jasność — BRIGHT oraz błyskanie — FLASH. Atrybuty nie są przypisane do poszczególnych punktów, gdyż metoda taka pociągnęłaby za sobą zajęcie dużej części pamięci. Jeżeli każdy punkt mógłby mieć jeden z ośmiu kolorów, to na pamięć atrybutów byłyby potrzebne 18432 bajty, ponieważ osiem kolorów można jednoznacznie określić za pomocą trzech bitów.

W ZX Spectrum atrybuty są nakładane na bloki o wymiarach 8 × 8 punktów, które są elementami macierzy 24 wiersze na 32 kolumny. Do każdego z elementów macierzy (bloku 8 × 8 punktów) jest przyporządkowany w polu atrybutów (ang. *attributes*) jeden bajt określający (rys. 11):

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
FLASH	BRIGHT	PAPER			INK		

Rys. 11. Organizacja bajtu atrybutów

FLASH (błyskanie kwadratu)	bit 7 = 1 błyskanie włączone bit 7 = 0 błyskanie wyłączone
BRIGHT (jasność, odcień koloru)	bit 6 = 1 jasność podwyższona bit 6 = 0 jasność normalna
PAPER (kolor tła)	bity 5, 4, 3 tło w jednym z ośmiu kolorów,
INK (kolor atramentu)	bity 2, 1, 0 atrament w jednym z ośmiu kolorów

Konsekwencją takiej organizacji pamięci obrazu w ZX Spectrum są ograniczone możliwości kolorowania obrazu. Przez kwadrat 8 × 8 punktów nie można przeprowadzić kilku różnokolorowych linii, gdyż przyjmą one kolor atramentu przyporządkowany do ostatnio kreślonej linii. W kwadracie takim można używać tylko jednego koloru atramentu oraz jednego koloru tła*).

Zawartość bajtu z dowolnego pola obszaru atrybutów (ATTR) można określić równaniem:

$$ATTR = 128*FLASH + 64*BRIGHT + 8*PAPER + INK$$

$$FLASH = \begin{cases} 1 \text{ włączony} \\ 0 \text{ wyłączony} \end{cases} : BRIGHT = \begin{cases} 1 \text{ podwyższona} \\ 0 \text{ normalna} \end{cases}$$

$$PAPER = 0, 1, 2, \ldots 7 \quad : INK = 0, 1, 2, \ldots 7$$

Informacje zawarte w tabl. 2 będą pomocne przy określaniu atrybutów dowolnego

*) Tym większy podziw należy mieć dla autorów niektórych gier komputerowych, którzy mając dość ograniczone możliwości potrafili tworzyć bardzo skomplikowane (pod względem struktury graficznej) i kolorowe obrazy.

Tablica 2. Znaczenie zawartości bajtu atrybutów

PAPER (tło)	INK (atrament) Black	Blue	Red	Magenta	Green	Yellow	Cyan	White	Uwagi
Black	0	1	2	3	4	5	6	7	Normalnie
	64	65	66	67	68	69	70	71	BRIGHT 1
	128	129	130	131	132	133	134	135	FLASH 1
	192	193	194	195	196	197	198	199	BRIGHT 1 + FLASH 1
Blue	8	9	10	11	12	13	14	15	Normalnie
	72	73	74	75	76	77	78	79	BRIGHT 1
	136	137	138	139	140	141	142	143	FLASH 1
	200	201	202	203	204	205	206	207	BRIGHT 1 + FLASH 1
Red	16	17	18	19	20	21	22	23	Normalnie
	80	81	82	83	84	85	86	87	BRIGHT 1
	144	145	146	147	148	149	150	151	FLASH 1
	208	209	210	211	212	213	214	215	BRIGHT 1 + FLASH 1
Magenta	24	25	26	27	28	29	30	31	Normalnie
	88	89	90	91	92	93	94	95	BRIGHT 1
	152	153	154	155	156	157	158	159	FLASH 1
	216	217	218	219	220	221	222	223	BRIGHT 1 + FLASH 1
Green	32	33	34	35	36	37	38	39	Normalnie
	96	97	98	99	100	101	102	103	BRIGHT 1
	160	161	162	163	164	165	166	167	FLASH 1
	224	225	226	227	228	229	230	231	BRIGHT 1 + FLASH 1
Cyan	40	41	42	43	44	45	46	47	Normalnie
	104	105	106	107	108	109	110	111	BRIGHT 1
	168	169	170	171	172	173	174	175	FLASH 1
	232	233	234	235	236	237	238	239	BRIGHT 1 + FLASH 1
Yellow	48	49	50	51	52	53	54	55	Normalnie
	112	113	114	115	116	117	118	119	BRIGHT 1
	176	177	178	179	180	181	182	183	FLASH 1
	240	241	242	243	244	245	246	247	BRIGHT 1 + FLASH 1
White	56	57	58	59	60	61	62	63	Normalnie
	120	121	122	123	124	125	126	127	BRIGHT
	184	185	186	187	188	189	190	191	FLASH
	248	249	250	251	252	253	254	255	BRIGHT 1 + FLASH 1

punktu na ekranie, szczególnie wtedy, gdy będą one wpisywane bezpośrednio do pamięci atrybutów za pomocą instrukcji POKE lub z poziomu języka asemblera.

W ZX Spectrum można także sterować kolorem granicznej części ekranu (ang. *Border*). Kolor brzegu zmieniać można z poziomu Basica za pomocą instrukcji BORDER *n*, gdzie *n* jest numerem odpowiadającym danemu kolorowi, podobnie jak przy określaniu INK i PAPER (zmienia się wtedy również kolor dwu dolnych linii obrazu). Innym sposobem zmiany koloru brzegu ekranu jest zmiana zawartości komórki pamięci przypisanej do jednobajtowej zmiennej systemowej o nazwie BORDCR i adresie 23624 (z poziomu Basica — BORDER *n* zmienia tę właśnie zmienną). Zmienna systemowa BORDCR określa także atrybuty dwu ostatnich linii ekranu (zwykle linie te nie są dostępne dla programującego w języku Basic). Znaczenie poszczególnych bitów tej zmiennej jest identyczne jak znaczenie bitów w polu atrybutów patrz rys. 11.

W oknie systemowym system operacyjny automatycznie określa kolor atramentu jako kontrastowy do koloru tła, niezależnie od tego jaki kolor atramentu i tła został określony za pomocą instrukcji INK*n*, PAPER*n* czy BORDER*n*. Zasada ta nie obowiązuje, gdy kolor atramentu i tła dwóch dolnych linii ekranu jest określony za pomocą polecenia POKE 23624, *n*. Po wykonaniu programu:

```
10 PAPER 5:INK 5: CLS
20 POKE 23624, 45
```

oraz naciśnięciu ENTER, cały ekran włącznie z brzegiem będzie koloru błękitnego, a komentarze i komunikaty o błędach będą niewidoczne, gdyż kolory atramentu i tła są identyczne. Powrót do pierwotnej sytuacji umożliwia wykonanie polecenia:

```
POKE 23624, 56
```

lub (prościej) instrukcji NEW.

Do zapamiętania kolorowego obrazu ZX Spectrum potrzebuje 6912 bajtów. Pamięć od adresu 16384 do 22527 jest przeznaczona na pamięć obrazu. Natomiast pamięć od adresu 22528 do 23295 włącznie jest zarezerwowana dla atrybutów.

Jak widać na rys. 11 kolor tła lub kolor atramentu są odczytywane na podstawie zawartości dwóch ciągów po trzy bity. Każdy z tych bitów może mieć wartość zero lub jeden. Z elementarnych zasad kombinatoryki wynika, że można otrzymać $2^3 = 8$ różnych ustawień poszczególnych bitów w danym ciągu. Jeżeli pierwszy bit (z prawej) tego ciągu ma wartość 1, a pozostałe są zerami, to odpowiada to zdefiniowaniu koloru niebieskiego. Analogicznie, jedynce na drugim bicie odpowiada kolor czerwony, a jedynce na trzecim bicie kolor zielony. Wszystkie pozostałe kolory (dostępne z klawiatury) — zgodnie z prawami przyrody — są mieszaniną trzech barw podstawowych. Jest to zilustrowane w tabl. 3.

ZX Spectrum może de facto używać nie ośmiu, lecz piętnastu kolorów. Osiąga się to definiując odpowiedni poziom jasności świecenia punktów na ekranie za pomocą instrukcji języka Basic — BRIGHT *n*. Jeżeli $n = 0$, to jasność jest normalna, a jeżeli $n = 1$, to jasność jest podwyższona. Czerń o jasności normalnej niczym nie różni się od czerni o jasności podwyższonej, dlatego istnieje możli-

Tablica 3 Zasady tworzenia kolorów

Kolor	Reprezentacja bitowa koloru			Odpowiadająca kolorowi wartość (dziesiętna)
	zielony green bit 2	czerwony red bit 1	niebieski blue bit 0	
Czarny black	0	0	0	0
Niebieski blue	0	0	1	1
Czerwony red	0	1	0	2
Fioletowy magenta	0	1	1	3
Zielony green	1	0	0	4
Błękitny cyan	1	0	1	5
Żółty yellow	1	1	0	6
Biały white	1	1	1	7

Uwaga: 1 — kolor włączony, 0 — wyłączony

wość wygenerowania 15 kolorów, a nie 16. Praktycznie niebieski o podwyższonej jasności różni się minimalnie od niebieskiego o jasności normalnej. Można więc powiedzieć, że jest 14 kolorów.

Najstarszy bit bajtu atrybutów określa migotanie (błyskanie). W Basicu migotanie ustawia się instrukcją FLASH *n*. FLASH 1 włącza je i wtedy w drukowanych na ekranie tekstach kolor tła zmienia się z koloru atramentu z określoną przez system częstotliwością. Po wykonaniu FLASH 0 efekt ten jest wyłączony.

W języku Basic ZX Spectrum istnieje funkcja ATTR (*w*, *k*), której argumentami są numer wiersza *w* (od 0 do 23) oraz numer kolumny *k* (od 0 do 31). Wartością tej funkcji jest zawartość bajtu atrybutów określonego współrzędnymi (*w*, *k*). Funkcję tę można zdefiniować samemu w następujący sposób:

DEF FN a(*w*, *k*) = PEEK (22528+w*32+k): REM = ATTR(*w*, *k*)

Wykonanie poniższego programu oraz analiza jego działania pomoże ocenić zrozumienie wiadomości o atrybutach.

```
  10 FOR i=22528 TO 22559: POKE
i,INT (RND*256): NEXT i
  20 PRINT
  30 INPUT "podaj nr kolumny  k=
";k
  40 IF k<0 OR k>=31 THEN GO TO
30
  50 PRINT ATTR (0,k): GO TO 30
```

Wykonanie 10 linii programu powoduje nałożenie losowych atrybutów na zerowy wiersz ekranu. Definiowane są pierwsze 32 bajty z pola atrybutów. Następnie (linia 20) ustawiana jest pozycja wydruku na ekranie. Pierwszy wydruk nastąpi w wierszu pierwszym. Komputer czeka teraz na wprowadzenie numeru kolumny, który nie może być mniejszy od 0 ani większy od 31 (linie 30 i 40). Po jego wprowadzeniu drukowana jest wartość odpowiedniego bajtu atrybutów z zerowego wiersza ekranu.

Drukowane na ekranie liczby można porównać z zawartością tablicy 2. Funkcję ATTR, użytą w tym programie, można zastąpić funkcją a(*w*, *k*) zdefiniowaną wyżej. Działanie programu nie powinno ulec zmianie.

* * Atrybuty a zmienne systemowe 3.3

Atrybuty w ZX Spectrum można definiować za pomocą instrukcji języka Basic. Tą drogą określa się:

— kolor tła — instrukcja PAPER *n*,
— kolor pisma — instrukcja INK *n*,
— odcień koloru — instrukcja BRIGHT *k*,
— migotanie — instrukcja FLASH *k*.

Argumentami PAPER oraz INK mogą być liczby od 0 do 9, natomiast argumentami BRIGHT oraz FLASH mogą być liczby 0, 1 i 8. Argumenty określające kolory o wartościach od 0 do 7 są zrozumiałe. Każdej liczbie jest przyporządkowany kolor zgodnie z tablicą 3. Argumenty 0 i 1 przy FLASH i BRIGHT zostały wcześniej omówione: 1 — włączony, 0 — wyłączony. Argument 8 może być użyty we wszystkich instrukcjach określających atrybuty. Oznacza on, że poprzedni atrybut ma zostać nie zmieniony. Inaczej mówiąc PAPER 8 oznacza ustawienie koloru „przezroczystego", spod którego będzie widać kolor poprzedni. Argument 9 może być użyty tylko z instrukcjami INK i PAPER. Użycie tego argumentu oznacza ustawienie kontrastowego koloru tła w stosunku do koloru atramentu i na odwrót.

Po włączeniu zasilania system operacyjny ustawia automatycznie biały kolor tła, czarny kolor atramentu oraz wyłącza BRIGHT i FLASH. Aktualne kolory oraz BRIGHT i FLASH włączony (wyłączony) są zapamiętywane przez system operacyjny w komórkach pamięci, zmiennych systemowych nazwanych ATTR P

oraz MASK P. Wykonanie każdej z podanych instrukcji zmieniających atrybuty powoduje zmianę zawartości tych zmiennych*).

Wartości ATTR P oraz MASK P ustawiane są w następujący sposób. Jeżeli argumentami instrukcji PAPER oraz INK są liczby z zakresu 0÷7 (oznaczające numer koloru), a instrukcji BRIGHT i FLASH 0 lub 1 (tzn. włączony lub wyłączony), to ustawiona zostaje zmienna ATTR P — bajt o adresie 23693. Zmienna ta ma strukturę identyczną do struktury bajtów mieszczących się w polu atrybutów (patrz rys. 11).
Programy:

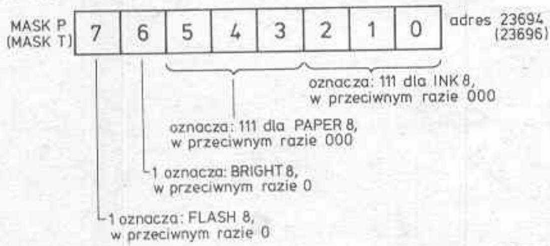
```
10 INK 2 : PAPER 6 : CLS
```

lub

```
10 POKE 23693, 50 : CLS
```

powinny dać jednakowy wynik.

Jeżeli parametrem instrukcji INK, PAPER, BRIGHT czy FLASH jest liczba 8, to ustawiana jest zawartość zmiennej MASK P (bajt o adresie 23694) w sposób pokazany na rys. 12. Jeśli zostanie wykonana np. instrukcja INK 8, to bity zmiennej MASK P przyporządkowane dla tego atrybutu będą miały wartość 1. Stanowi



Rys. 12. Organizacja zmiennej systemowej MASK P (MASK T)

to informację dla systemu operacyjnego, że w czasie wykonywania, wydruków na ekranie informacje o atrybutach (w tym wypadku INK) należy pobierać z pamięci ekranu, a nie jak to ma zwykle miejsce ze zmiennej ATTR P. Jeżeli wykonana zostanie jedna z instrukcji: PAPER, INK, BRIGHT, FLASH, INVERSE, OVER, to interpreter zmieni wartość zapamiętaną w zmiennych MASK P i ATTR P.

W czasie pisania tekstu na ekranie atrybuty znaków wprowadzanych na ekran są wytwarzane na podstawie zawartości tych zmiennych. Nazwano je *atrybutami globalnymi*, dotyczą bowiem wszystkich instrukcji PRINT. Jeśli natomiast PAPER, INK itd. zostanie użyte jako opcja instrukcji PRINT (tzn. po słowie kluczowym PRINT), to atrybuty dotyczą tylko tej instrukcji i dlatego nazwano je *atrybutami chwilowymi* lub *lokalnymi*.

*) Informacje podane w dalszej części punktu 3.3 są potrzebne głównie podczas programowania w języku asemblera.

Jeżeli zostaną zdefiniowane nowe atrybuty globalne, to nie oznacza to bynajmniej, że zmienią się kolory na ekranie. Na przykład, choć zmieniamy kolor tła instrukcją PAPER 5 z białego na błękitny, ekran pozostanie biały, zmienia się natomiast zawartość zmiennej ATTR P. Dopiero wykonanie:

PRINT "dowolny tekst"

spowoduje, że linie ekranu, w których zostanie wydrukowany „dowolny tekst", będą miały niebieskie tło, gdyż po napisaniu wybranych znaków na ekranie zmienione zostaną atrybuty, zgodnie z zawartością bajtu o adresie 23693.

Kolor ekranu można zmienić, jeżeli zostaną wykonane instrukcje:

PAPER 5 : CLS

Spowodują one wpisanie w bajcie pamięci odpowiedzialnym za atrybuty globalne odpowiedniej wartości. Następnie instrukcja CLS „czyści" pamięć obrazu i definiuje nowe atrybuty globalne zgodnie z zawartością komórki pamięci o adresie 23693. Instrukcja CLS zawsze korzysta ze zmiennej systemowej ATTR P, np. wykonanie

BRIGHT 1 : INK 2 : PAPER 6 : CLS

jest równoważne

POKE 23693, 114 : CLS

W ZX Spectrum istnieje możliwość określenia nie tylko atrybutów globalnych, ale i lokalnych. Atrybuty lokalne dotyczą tylko jednej instrukcji PRINT. Ilustruje to program:

```
  10 PAPER 1: INK 7: CLS
  20 PRINT "atrybuty globalne"
  30 PRINT PAPER 6; INK 2; BRIGH
T 1; FLASH 1;"atrybuty lokalne"
  40 PRINT "atrybuty globalne"
```

Jako atrybuty globalne zdefiniowano niebieski kolor tła oraz biały kolor atramentu. FLASH i BRIGHT są wyłączone. W linii 30 programu są zdefiniowane atrybuty lokalne. Obowiązują one tylko w tej linii programu. Po wykonaniu linii 40 tekst „atrybuty globalne" jest drukowany na ekranie zgodnie z atrybutami zdefiniowanymi w linii 10.

Wartości chwilowe atrybutów są zapamiętywane w zmiennej systemowej ATTR T o adresie 23695. Bajt pamięci przyporządkowany ATTR T ma identyczną strukturę jak ATTR P — rys. 11. Po załączeniu zasilania do obu zmiennych ATTR jest wpisywana liczba 56, co odpowiada użyciu czarnego atramentu i białego tła przy wyłączonych BRIGHT i FLASH.

Gdy użyta zostanie w instrukcji PRINT jedna z opcji: INK, PAPER, BRIGHT itd. z argumentem równym 8, to ustawiona zostanie zawartość jednobajtowej zmiennej systemowej MASK T, która jest podobna do MASK P. Jedyną różnicą między nimi jest to, że MASK T określa atrybuty o argumencie 8, jako

chwilowe, a nie globalne (widać tu analogię do ATTR P i ATTR T). Wartość MASK T jest pamiętana w komórce pamięci o adresie 23696. Organizacja tej komórki pamięci jest przedstawiona na rys. 12. Ilustruje to następujący program:

```
 10 PRINT PEEK 23696
 20 FLASH 8: BRIGHT 8: PAPER 8:
INK 8
 30 PRINT PEEK 23696
```

Początkowo w komórce pamięci o adresie 23696 wszystkie bity mają wartość zero. Po wykonaniu linii 20 programu pod adres MASK T są wpisywane same jedynki, czyli 255.

Ostatnią zmienną systemową pomagającą systemowi operacyjnemu ZX Spectrum w określaniu atrybutów globalnych i lokalnych (choć nie tylko) jest P FLAG — bajt o adresie 23697. Zmienna ta zawiera w poszczególnych bitach znaczniki używane podczas tworzenia wydruków na ekranie. Jeżeli PAPER ma argument 9 (rys. 13), to w zależności od tego czy odnosi się to do całego ekranu, czy też jest zdefiniowany lokalnie, bit 7 lub bit 6 mają odpowiednio

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
globalny PAPER 9	chwilowy PAPER 9	globalny INK 9	chwilowy INK 9	globalny INVERSE 1	chwilowy INVERSE 1	globalny OVER 1	chwilowy OVER 1	P FLAG (23697)

Rys. 13. Organizacja zmiennej systemowej P FLAG

wartość jeden. Globalnemu i lokalnemu INK 9 odpowiadają bity 5 i 4. Bity od 0 do 3 zmiennej systemowej P FLAG są znacznikami globalnych i lokalnych INVERSE 1 oraz globalnych i lokalnych OVER 1 (te instrukcje języka Basic zostaną bliżej omówione w p. 3.4.1). Globalnemu i lokalnemu INVERSE 1 odpowiadają bity 3 i 2. Natomiast globalnemu i lokalnemu OVER 1 odpowiadają bity 1 i 0.

* Tryby graficzne ZX Spectrum 3.4

Tryb znakowy 3.4.1

Każdy znak pisany na ekranie za pomocą języka Basic jest określony w polu 8×8 punktów. Dlatego z punktu widzenia interpretera Basica ekran może być macierzą zawierającą 24 wiersze na 32 kolumny. Taki tryb, w którym do tworzenia obrazów używa się znaków, a nie punktów ekranu, nazywa się *trybem znakowym*. ZX Spectrum de facto nie ma trybu znakowego. „Prawdziwy" tryb znakowy taki jaki jest używany np.: w Commodore 64 czy Atari 800 XL polega na tym, że w pamięci ekranu zapamiętywane są kody znaków zamiast informacji

określających każdy punkt ekranu. Układ sterujący wyświetlaniem obrazu na ekranie (odpowiednik ULA) na podstawie kodów znaków generuje obraz. Tymczasem w ZX Spectrum pamięć ekranu zawiera zawsze informacje dla każdego punktu na ekranie — w relacji jeden bit — jeden punkt. Jednakże za pomocą instrukcji języka Basic można pisać na ekranie znakami np.:

PRINT "to jest tryb znakowy"

Odpowiednie programy umieszczone w systemie operacyjnym pobierają kolejne znaki z instrukcji PRINT i na podstawie kodu znaku odszukują jego adres w generatorze znaków, tj. w obszarze pamięci, w którym zapamiętano kształty wszystkich znaków. Następnie osiem bajtów definiujących kształt każdego znaku jest umieszczanych w odpowiednim miejscu pamięci ekranu. W związku z istnieniem możliwości wypełniania pamięci obrazu generowanego przez ZX Spectrum za pomocą znaków (a nie tylko pojedynczych punktów) taki tryb graficzny nazywa się potocznie trybem znakowym. Tu także nazwa ta będzie używana. Za pomocą znaków można także tworzyć rysunki.

Uwagi o pracy w trybie znakowym

Jak już zostało powiedziane w trybie znakowym ekran telewizora jest podzielony na 32*24 = 768 elementów. Górne 22 wiersze ekranu przeznaczone są na:
— wydruki użytkownika,
— wydruki treści programów — tzw. listingi programów.
Natomiast dolne dwa wiersze, tzw. o k n o s y s t e m o w e, są wykorzystywane:
— do wprowadzania z klawiatury linii programów do pamięci mikrokomputera (nieraz jedna linia zajmuje więcej niż dwa wiersze, wówczas zachodzi na górną część ekranu bezpośrednio nad oknem systemowym),
— do wprowadzania danych (dotyczy to instrukcji INPUT),
— na komunikaty systemu np.: "No room for line" (brak w pamięci komputera miejsca na nową linię programu), czy też "start tape, then press any key" (włącz magnetofon, a następnie naciśnij dowolny klawisz) i inne,
— w trybie edycji do poprawiania i uzupełniania wybranych linii programu,
— przez użytkownika, który może te wiersze wykorzystać choćby w następujący sposób (patrz rozdz. 4):

```
PRINT#0: "Jestem w oknie systemowym": PAUSE 100
PRINT#1: "i ja również" : PAUSE 100
```

Jeżeli w powyższych poleceniach nie byłoby instrukcji PAUSE 100, to napisany w oknie systemowym tekst ulegałby natychmiastowemu zatarciu przez informację o poprawnym zakończeniu programu — komunikat systemu OK 0 : 1. Natomiast PAUSE 100 pozwoli go oglądać przez okres około 2 s.

O tym, ile linii jest przydzielonych do dolnej części ekranu mówi jednobajtowa zmienna systemowa zwana DF SZ o adresie 23659. Zwykle w tym miejscu pa-

pamięci jest umieszczona przez system operacyjny liczba 2. Liczbę linii w oknie systemowym można zmienić rozkazem:

POKE 23659, *liczba linii*

gdzie liczba linii może być od 0 do 23.
Ilustruje to program:

```
  10 POKE 23659,0
  20 FOR i=0 TO 23: PRINT i: NEX
T i
  30 PAUSE 0
  40 POKE 23659,2
```

Zmian przydziału liczby linii w oknie systemowym należy dokonywać bardzo rozważnie. W przeciwnym razie można doprowadzić do błędnego działania systemu (o czym ostrzega producent) lub spowodować zawieszenie się programu i utratę zawartych w pamięci informacji.

Wiersze i kolumny w trybie znakowym są numerowane od 0 tak, że wiersz 24 ma w rzeczywistości numer 23, a kolumna 32 ma w rzeczywistości numer 31. Numeracja wierszy i kolumn rozpoczyna się w lewym górnym rogu ekranu. Każdemu znakowi na ekranie można przyporządkować współrzędne na tej samej zasadzie jak elementom tablicy dwuwymiarowej. Zgodnie z tym znak górnego lewego rogu ma współrzędne (0, 0).

Basic a tryb znakowy

W Basicu ZX Spectrum istnieją instrukcje pozwalające na pracę w trybie znakowym. Najważniejsza z nich jest instrukcja PRINT, która została omówiona w rozdz. 2. Pozwala ona na dokonywanie wydruków na ekranie telewizora czy monitora. Jeżeli chce się, aby wydruk rozpoczynał się w wybranym miejscu ekranu należy skorzystać z opcji AT y, x. Argumentami jej są numer wiersza y oraz numer kolumny x. Opcja ta pozwala na drukowanie dowolnych znaków w wybranym przez użytkownika miejscu w górnej części ekranu (w górnych 22 liniach). Na przykład, wykonanie programu:

10 PRINT AT 0, 0; "P"; AT 21, 31; "K"

spowoduje napisanie w lewym górnym rogu ekranu litery P oraz w prawym dolnym rogu ekranu litery K. Po powtórnym użyciu tych samych wartości argumentów w funkcji AT nastąpi skasowanie starego znaku i zastąpienie go nowym, tak jak w programie:

```
  20 PRINT AT 20,15;"o"; OVER 1;
AT 10,15;"´"
  30 PAUSE 150
  40 PRINT AT 20,13;"kr";AT 10,1
6;"l"
```

Zatrzymanie programu następuje przy jednoczesnym naciśnięciu CAPS SHIFT oraz BREAK.

W niektórych przypadkach korzystne jest ominięcie powyższej reguły dotyczącej wymazywania starego znaku przez nowy. W Basicu ZX Spectrum możliwe jest nadrukowywanie jednego znaku na drugi za pomocą instrukcji OVER *n*. Argumentami tego polecenia może być 0 lub 1. Rozkaz OVER 1 włącza specjalny rodzaj nadrukowywania, natomiast OVER 0 wyłącza go. Wówczas, jeżeli jakiś znak jest pisany w miejscu już zajętym, to nie następuje usunięcie starego znaku, lecz jedynie dopisanie tych punktów, które mają w nowym znaku kolor atramentu. Miejsca, w których punkty w kolorze atramentu nałożonych na siebie znaków pokrywają się, tracą swą barwę i przyjmują kolor tła, co stanowi (zdaniem autorów) pewną wadę. Działanie OVER 1 ilustruje program:

```
10 PRINT AT 10, 15; "X"; CHR$ 8; OVER 1; "H"
```

W wyniku wykonania programu otrzymamy znak, który jest sumą (w wyżej określonym sensie) znaków X oraz H. Wykonanie CHR$ 8 jest równoważne cofnięciu pozycji wydruku o jedno miejsce.

Instrukcja (lub opcja) OVER 1 może być przydatna przy tworzeniu znaków złożonych. Przykładem takiego znaku jest polska litera „Ó". Można ją napisać w następujący posób:

```
10 PRINT AT 11,15;"stare znaki
": PAUSE 100
20 PRINT AT 11,15;"nowe znaki
": PAUSE 100
30 GO TO 10
```

Zadanie odwrotne do PRINT AT y, x realizuje funkcja SCREEN$ (y, x). Testuje ona określoną argumentami (y — numer wiersza, x — numer kolumny) pozycję na ekranie (w górnej jego części) i daje w wyniku znak zapisany w tym miejscu. Funkcja ta, podobnie jak opcja AT, nie dotyczy okna systemowego. Dla lepszego zrozumienia jej działania należy wykonać program:

```
10 PRINT AT 7, 6; "TEST jak T"; PAUSE 50
20 PRINT AT 7, 15; SCREEN$ (7, 6)
```

W wyniku wykonania programu na ekranie zostanie napisane „TEST jak T".

Wartością funkcji SCREEN$ (y, x) może być znak używany przez komputer, tzn. zdefiniowany w generatorze znaków (patrz p. 3.7). Jeżeli w testowanym przez SCREEN$ (y, x) polu znajdzie się nieistniejący znak, to wartością tej funkcji jest tekst pusty. Można to pokazać w prosty sposób, wykorzystując instrukcję DRAW, opisaną dalej

```
10 PLOT 247,167 : DRAW 8,8
20 PRINT AT 0,0: SCREEN$ (21,31)
```

lub

```
30 PRINT AT 21,31; "A"
40 PRINT 21,0; SCREEN$ (21,31)
```

W linii 10 za pomocą instrukcji DRAW został narysowany znak nieużywany przez interpreter Basica ZX Spectrum. Funkcja SCREEN$ „nie zauważy" takiego znaku i dlatego po wykonaniu linii 20 nic się nie wydrukuje. Linie 30 i 40 ilustrują, że funkcja SCREEN$ odczytuje z ekranu jedynie znak o kształcie zdefiniowanym w generatorze znaków ZX Spectrum.

W trybie znakowym do ustalania pozycji wydruku w obrębie wybranego wiersza używa się opcji TAB *n*, gdzie *n* jest numerem kolumny, od której należy rozpocząć wydruk. TAB *n* działa w ten sposób, że dopisuje znaki odstępu (spacje), aż do momentu ustawienia miejsca wydruku w kolumnie *n*. Argument *n* może być liczbą rzeczywistą dodatnią z przedziału (0, 65535). Mikrokomputer jako argument TAB bierze tylko część całkowitą wprowadzonej liczby. Jeżeli *n* będzie liczbą spoza podanego zakresu, to na ekranie zostanie wyświetlony komunikat o błędzie. ZX Spectrum używa wartości argumentu TAB modulo 32. Oznacza to, że numer kolumny, w której ma nastąpić wydruk, ustawiony przez TAB, jest całkowitą resztą z dzielenia *n* przez 32. Zgodnie z tym zapis:

```
PRINT "A"; TAB 10; "B"
```

jest równoważny

```
PRINT "A"; TAB 42.4; "B"
```

W związku z tym w praktyce jako argumentu TAB używa się liczb z przedziału od 0 do 31.

Zdarza się, że w jednej instrukcji PRINT używa się kilku opcji TAB *n*. Jeżeli po kolejnym TAB *n* wydruk ma nastąpić w miejscu już zajętym przez znak, to komputer obniża pozycję wydruku o jeden wiersz, tak jak w programie:

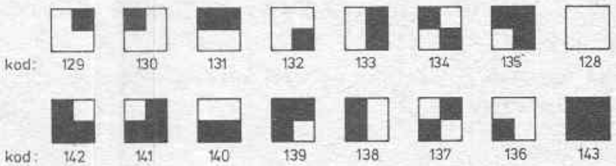
```
10 PRINT "ABC"; TAB 20; "DEFG"; TAB 21; "HIJK"
```

Dzieje się to więc inaczej niż przy użyciu opcji AT y,x.

Podczas dokonywania wydruków na ekranie czy drukarce często zachodzi potrzeba wyróżnienia wybranej informacji spośród innych, choćby w celu podkreślenia jej ważności. Jedną z metod rozwiązania tego problemu jest użycie instrukcji (opcji) INVERSE *n*, gdzie *n* jest 1 lub 0. INVERSE z argumentem 1 oznacza, że w wydruku kolor atramentu będzie zamieniony z kolorem tła. Obraz będzie negatywem swej pierwotnej postaci. Negatywowy obraz może być określony globalnie na całym ekranie lub też lokalnie — w instrukcji PRINT. Wyłączenie trybu INVERSE następuje przez wykonanie instrukcji INVERSE 0. Działanie INVERSE *n* ilustruje program:

```
  10 INVERSE 1
  20 PRINT INVERSE 0;AT 0,10;"IN
VERSE 0"
  30 PRINT AT 21,10;"INVERSE 1"
```

Podstawowym narzędziem do tworzenia w trybie znakowym grafiki na ekranie telewizora są znaki graficzne dostępne z klawiatury w trybie graficznym (w trybie G). ZX Spectrum ma 16 własnych symboli graficznych. Każdy znak graficzny jest określony w polu 8×8 punktów, czyli w polu jednego znaku w ZX Spectrum. Wszystkie symbole graficzne omawianego mikrokomputera drukuje na ekranie telewizora w dwu kolumnach program (patrz rys. 14):



Rys. 14. Obraz znaku w trybie G

```
  10 FOR i=128 TO 143 STEP 2
  20 PRINT CHR$ i,CHR$ (i+1): PR
INT
  30 NEXT i
```

Pierwszy znak graficzny znajdujący się w lewym górnym rogu ekranu jest niewidoczny. Jest to tzw. spacja graficzna. Znak ten ma kod równy 128. Aby przekonać się, że znak ten naprawdę istnieje należy wykonań polecenie:

PRINT PAPER 6; CHR$ 128

Negatywem spacji graficznej jest znak o kodzie 143.

Jak ZX Spectrum tworzy kody swoich znaków graficznych? Jest to bardzo proste. Każdy kwadrat 8×8 pixeli jest podzielony na cztery mniejsze kwadraty 4×4 punkty. Każdemu z tych kwadratów przyporządkowany jest bit.

Kod znaku jest równy 128+1*f(bit 0)+2*f(bit 1)+4*f(bit 2)+8*f(bit 3), gdzie funkcja f(bit i) przyjmuje wartości zero lub jeden. Biorąc to pod uwagę znak „■" powinien mieć kod

Kod (■) = 128+1*1+2*1+4*0+8*1 = 139

Stąd, po wykonaniu instrukcji:

PRINT CHR$ 139

omawiany znak zostanie wydrukowany na ekranie.

Rysowanie na ekranie w trybie znakowym ZX Spectrum demonstruje prosty program: „Zaczarowany ołówek". Program ten używa sześciu klawiszy sterujących. Cztery z nich to tzw. kursory (5, 6, 7, 8). Sterują one ruchem w kierunku zgodnym z oznaczeniami na danym klawiszu. Naciśnięcie klawisza „0" włącza „gumkę". Można ją przesuwać po ekranie za pomocą kursorów. Gumka

```
  10 REM zaczarowany olowek
  20 LET x=INT (RND*32): LET y=I
NT (RND*22): LET a$=CHR$ 143
  30 IF INKEY$="1" THEN LET a$=C
HR$ 143
  40 IF INKEY$="0" THEN LET a$=C
HR$ 128
  50 PRINT AT y,x;a$
  60 LET x=x-(INKEY$="5" AND x>0
)+(INKEY$="8" AND x<31)
  70 LET y=y-(INKEY$="7" AND y>0
)+(INKEY$="6" AND y<21)
  80 IF INKEY$="" THEN GO TO 60
  90 GO TO 30
```

wyciera wszystko co spotyka na swej drodze. Opuszczenie pisaka następuje po naciśnięciu klawisza „1".

Interpunkcja

Ważnymi elementami instrukcji PRINT czy INPUT są znaki interpunkcyjne takie jak średnik (;) czy przecinek (,). Wykonanie polecenia

PRINT "KARO", "LINA"

lub

PRINT "BUTTER", "FLY"

nie jest równoważne

PRINT "KARO"; "LINA"

lub

PRINT "BUTTER"; "FLY"

W pierwszym przypadku wyrazy są oddzielone od siebie przecinkiem, a w drugim średnikiem. Przecinek powoduje przesunięcie wydruku do następnej pozycji tabulacji*). Średnik powoduje, że następny wydruk rozpocznie się bezpośrednio po poprzednim.

Znaczenie interpunkcji obrazuje poniższy program:

```
  10 FOR i=0 TO 9: PRINT i: NEXT
 i
  20 PAUSE 0: CLS
  30 FOR i=0 TO 9: PRINT i,: NEX
T i
  40 PAUSE 0: CLS
  50 FOR i=0 TO 9: PRINT i;: NEX
T i
```

*) Pozycje tabulacji (ustalone miejsca wydruku w wierszu) ustawione są w kolumnie 0 i 16. Jeżeli tekst pisany od początku wiersza ekranu jest zakończony przecinkiem, to następny wydruk rozpocznie się od 16 kolumny w tym wierszu o ile tekst nie był dłuższy niż 16 znaków. W takim przypadku nastąpi zmiana wiersza.

Instrukcja PAUSE 0 w linii 20 i 40 powoduje zatrzymanie wykonywania programu do momentu naciśnięcia dowolnego klawisza. Instrukcja CLS kasuje ekran.

Kolejnym znakiem interpunkcyjnym używanym przez Basic ZX Spectrum jest apostrof ('). Umieszczenie apostrofu w instrukcji PRINT, INPUT czy LP-RINT (patrz 4.7) jest równoważne poleceniu „zacznij wydruk w nowej linii". Praktyczne działanie apostrofu (apostrofów) ilustruje program:

```
  10 FOR i=0 TO 21: PRINT AT i,1
3;i: NEXT i
  20 PRINT PAPER 5;AT 10,16;"WIE
RSZ NR 10";AT 12,16;"WIERSZ NR 1
2"
  30 PRINT PAPER 6;AT 9,0;'"wier
sz nr 10"''"wiersz nr 12"
```

Po dokonaniu numeracji wierszy na ekranie, wykonanie linii 20 powyższego programu spowoduje wydruk tekstów „WIERSZ NR 10" oraz „WIERSZ NR 12" w ustalonych za pomocą opcji AT (opisanej wcześniej) miejscach ekranu. Linia 40 realizuje wydruk tekstów „wiersz nr 10" i „wiersz nr 12" dokładnie w tych samych wierszach ekranu co linia 20. Należy zwrócić uwagę, że pierwsze argumenty AT (linia 20 i 40), występujące bezpośrednio po instrukcji PRINT, są różne. Kolumna, w której ma nastąpić wydruk, jest ustalana opcją AT lub TAB. Stąd wniosek, że każdy apostrof zmienia pozycję wydruku o jeden wiersz w dół.

Apostrofu można także używać do odpowiedniego formatowania wydruków na ekranie telewizora. Ilustruje to następujący program:

```
  10 PRINT "Szlachetne zdrowie,"
'"nikt sie nie dowie,"'"jako sma
kujesz,"'"az sie zepsujesz."'"Ta
m czlowiek prawie"'"widzi na jaw
ie..."
```

Kasowanie ekranu i części ekranu

Jak już zostało wcześniej powiedziane, instrukcja CLS kasuje ekran (patrz p. 3.2), czyli po jej wykonaniu zostają zgaszone wszystkie punkty ekranu. Nie jest to jedyne zastosowanie CLS. Często polecenia tego używa się do zmiany na całym ekranie wcześniej zdefiniowanych atrybutów. Ilustruje to następujący program:

```
10 PAPER 6:FLASH 1:BRIGHT 1:CLS:PRINT AT 11,16;"B"
```

W wielu praktycznych przypadkach istnieje konieczność skasowania tylko części ekranu. Jedną z metod rozwiązania tego problemu jest drukowanie znaków spacji w instrukcji PRINT w następujący sposób:

```
  10 FOR i=1 TO 704: PRINT "£";:
 NEXT i: REM zapełnienie całego
ekranu znakiem £"
  20 PAUSE 100
  30 FOR i=0 TO 21: PRINT AT i,1
6;"                ": REM w cudz
ysłowiu jest 16 'spacji'"
  40 NEXT i
  10 FOR i=1 TO 704: PRINT "£";:
 NEXT i: REM zapełnienie całego
ekranu znakiem £"
  20 PAUSE 100
  30 FOR i=5 TO 16: PRINT AT i,1
0;"            ": REM w cudzysło
wiu jest 12 'spacji'"
  40 NEXT i
```

Wykonanie pętli ze zmienną sterującą *k* powoduje skasowanie części ekranu. W tym przypadku skasowana zostanie prawa połowa ekranu. Dobierając odpowiednio wartość początkową i końcową zmiennej sterującej pętli *k* oraz liczbę spacji w cudzysłowie można kasować dowolną część ekranu. Dla przykładu można zmienić linię 30 i 40 powyższego programu:

```
  10 FOR i=1 TO 704: PRINT "£";:
 NEXT i: REM zapełnienie całego
ekranu znakiem £"
  20 PAUSE 100
  30 FOR i=5 TO 16: PRINT AT i,1
0;"            ": REM w cudzysło
wiu jest 12 'spacji'"
  40 NEXT i
```

Tak zmodyfikowany program będzie usuwał wszystkie znaki z prostokąta umieszczonego na środku ekranu.

Część ekranu, począwszy od dołu, można także skasować za pomocą instrukcji INPUT omówionej w poprzednim rozdziale. Robi się to w następujący sposób:

```
  10 FOR k=0 TO 21: PRINT k: NEX
T k
  20 INPUT "ile wierszy skasowac
?";n
  30 PRINT AT 0,0
  40 INPUT AT n+1,0
```

Za pomocą opcji AT 0,0 następuje ustawienie aktualnej pozycji wydruku w lewym górnym rogu ekranu. Natomiast linia 40 kasuje *n* dolnych wierszy. W linii 40 przyjęto argument opcji AT zwiększony o 1. Dzieje się tak dlatego, że INPUT obejmuje swoim działaniem wiersze okna systemowego, w tym przypadku pierwszy wiersz tego okna. Czy wykonanie powyższego programu bez linii 30 coś zmieni?

Tak, i będą to zmiany istotne. Po usunięciu linii 30 skasowane zostanie 10 górnych linii ekranu, a pozostałe 12 przesunie się do góry. Takie działanie programu związane jest z tym, że linia 40 powiększa tylko okno systemowe. Jeżeli aktualna pozycja wydruku znajduje się tuż nad nim, to relacja ta się nie zmienia, a powiększenie okna systemowego następuje kosztem *n* górnych linii ekranu. Przy ustawieniu aktualnej pozycji wydruku w lewym górnym rogu okno systemowe powiększa się kosztem miejsca na ekranie znajdującego się bezpośrednio nad nim.

Przesuwanie ekranu w trybie znakowym

Ekran w trybie znakowym jest traktowany jako macierz 24 × 32 elementy. Górna część ekranu (bez okna systemowego) jest macierzą składającą się z 22 × 32 = = 704 kwadratów — pól znakowych. Tą część ekranu można więc potraktować jako łańcuch składający się z 704 znaków. Teraz przesuwanie ekranu będzie mogło być realizowane tak jak operacja na łańcuchu znaków. Proste procedury przesuwania ekranu można w języku Basic napisać w następujący sposób:

- Przesuwanie wydruku na ekranie do góry:

```
  10 DIM e$(704): DIM a$(32)
  20 INPUT "wprowadz dowolny cia
g znakow nie dluzszy niz 704 zna
ki";e$
  30 PRINT AT 0,0;e$
  40 LET e$=e$(33 TO )+a$
  50 GO TO 30
```

- Przesuwanie w dół:

```
  10 DIM e$(704)
  20 INPUT "wprowadz dowolny cia
g znakow nie dluzszy niz 704 zna
ki";e$
  30 PRINT AT 0,0;e$
  40 LET e$="
                "+e$( TO 672): REM
 w cudzyslowiu sa 32 'spacje'"
  50 GO TO 30
```

- Przesuwanie w lewo:

```
  10 DIM e$(704)
  20 INPUT "wprowadz dowolny cia
g znakow nie dluzszy niz 704 zna
ki";e$
  30 PRINT AT 0,0;e$
  40 FOR i=1 TO 673 STEP 32
  50 LET e$(i TO i+31)=e$(i+1 TO
 i+31)+" ": REM w cudzyslowiu je
st jedna 'spacja'"
  60 NEXT i
  70 GO TO 30
```

• Przesuwanie w prawo:

```
  10 DIM e$(704)
  20 INPUT "wprowadz dowolny cia
g znakow nie dluzszy niz 704 zna
ki";e$
  30 PRINT AT 0,0;e$
  40 FOR i=1 TO 673 STEP 32
  50 LET e$(i TO i+31)=" "+e$(i
TO i+30): REM w cudzyslowiu jest
 jedna 'spacja'"
  60 NEXT i
  70 GO TO 30
```

Informacje zawarte w znikającym z ekranu wierszu lub kolumnie traci się; zostają one zastąpione spacjami. Można to zmienić choćby tak, że znikający wiersz czy kolumnę dopisuje się na koniec przesuwanego łańcucha znaków np.:

• Przesuwanie do góry:

```
  10 DIM e$(704)
  20 INPUT "wprowadz dowolny cia
g znakow nie dluzszy niz 704 zna
ki";e$
  30 PRINT AT 0,0;e$
  40 LET e$=e$(33 TO )+e$( TO 33
)
  70 GO TO 30
```

• Przesuwanie w lewo:

```
  10 DIM e$(704)
  20 INPUT "wprowadz dowolny cia
g znakow nie dluzszy niz 704 zna
ki";e$
  30 FOR i=1 TO 673 STEP 32
  40 PRINT AT 0,0;e$
  50 LET e$(i TO i+31)=e$(i+1 TO
 i+31)+e$(i TO i+31)
  60 NEXT i
  70 GO TO 30
```

Otrzymane w wyniku działania dwóch ostatnich programów obrazy będą zachowywały się tak, jakby były napisane na rolce (walcu). Obraz będzie rolowany. Tekst, który będzie znikał z góry ekranu będzie pojawiał się na dole, a tekst, który będzie znikał z lewej strony ekranu będzie pojawiał się z jego prawej strony.

Przesuwanie (rolowanie) obrazu w lewo lub w prawo jest znacznie wolniejsze od przesuwania (rolowania) obrazu w górę czy w dół. Programy na szybkie przesuwanie (rolowanie) obrazu i to z dokładnością do jednego punktu ekranu muszą być pisane w kodzie maszynowym. Wówczas obraz przesuwałby się szybko i płynnie.

W pewnych programach (np. grach) zachodzi niekiedy potrzeba opisania bieżącej sytuacji na ekranie bez niszczenia aktualnego obrazu. W takim przypadku na opis przeznaczyć można jeden wiersz okna systemowego. Jeżeli opis jest dłuższy niż 32 znaki, to tekst na ekranie można przesuwać znak po znaku w pierwszym wierszu okna systemowego tak jak np. w programie:

```
  10 REM cos jeszcze o przewijan
iu czesci ekranu
  20 DIM p$(32)
  30 LET t$=p$+"tekst znajdujacy
 sie w cudzyslowiu w linii 30 te
go programu bedzie wyswietlany n
a ekranie w odpowiednim wierszu"
+p$
  40 GO SUB 100
  50 LET t$=p$+"liczba wyswietla
nych w ten sposob napisow moze b
yc dowolna"+p$
  60 GO SUB 100
  70 STOP
 100 REM procedura wydruku
 110 FOR k=1 TO LEN t$-31
 120 PRINT #0;AT 1,0; PAPER 1; I
NK 6;t$(k TO k+31): PAUSE 12
 130 NEXT k
 140 RETURN
```

Istotną rzeczą w programie jest tablica znakowa P$. Instrukcja DIM P$ (32) rezerwuje miejsce na 32 znaki. Po zadeklarowaniu P$ zawiera ona 32 spacje, np. po wykonaniu polecenia:

DIM P$ (32): PRINT PAPER 5; P$

na ekranie zostanie wydrukowany wiersz składający się z 32 odstępów o błękitnym kolorze tła. Dzięki temu, że drukowany tekst rozpoczyna się i kończy znakami z tablicy P$ (a więc spacjami) jest on bardziej czytelny, nie pojawia się on bowiem na ekranie, lecz „wsuwa" od lewej jego strony. Szybkość wyświetlania kolejnych znaków reguluje ostatnia w linii 120 instrukcja PAUSE 10.

Jeżeli w linii 120 usunie się pierwsze trzy znaki występujące po słowie kluczowym PRINT, tzn. „#0;", wówczas tekst będzie wyświetlany w drugim od góry wierszu ekranu. Wybór dowolnego wiersza górnej części ekranu następuje po zmianie pierwszego argumentu funkcji AT.

Wydruki liczb zmiennoprzecinkowych

Często istotne jest, aby wydruki były czytelne i miały estetyczny wygląd. Dotyczy to szczególnie zobrazowania wyników obliczeń, czy danych liczbowych. W związku z tym zostaną podane programy ustalające formę wydruku zgodnie z określonymi przez użytkownika wymogami.

Wyrównanie wydruku do lewej krawędzi ekranu nie przedstawia żadnego problemu. Realizuje się je za pomocą instrukcji PRINT. Podobnie realizuje się równanie wydruków do wybranej kolumny ekranu. W tym przypadku są pomocne opcje: AT y,x lub TAB x, ustalające miejsce wydruku w żądanej kolumnie.

Każdą liczbę, która ma zostać wydrukowana, można za pomocą funkcji STR$ zamienić na łańcuch znaków:

LET a$ = STR$ (*liczba gotowa do wydruku*)

Po takiej operacji zmienna prosta staje się łańcuchem znaków.

Niżej zostaną podane programy realizujące różne formy wydruku:

• Wyrównanie do prawej krawędzi ekranu:

```
9000 PRINT TAB 32-LEN (STR$ (lic
zba));liczba: RETURN
9010 PRINT AT n,32-LEN (STR$ (li
czba));liczba: RETURN
```

Procedura w linii 9000 drukuje liczbę w kolejnym wierszu. Natomiast procedura w linii 9010 pozwala na wybranie dowolnego wiersza, w którym ma nastąpić wydruk. Jeżeli w obu programach liczbę 32, występującą jako argument TAB i AT, zastąpi się inną liczbą całkowitą z przedziału od 0 do 32, wówczas procedury te będą wyrównywały wydruki do kolumny ekranu określonej nowym argumentem. Nowy argument TAB i AT (drugi) co do wartości nigdy nie może być mniejszy od długości łańcucha STR$ *liczba*, czyli:

nowy argument > LEN STR$ (*liczba*)

Ogólne procedury na wyrównanie wydruku do wybranej kolumny będą miały postać:

```
9020 PRINT TAB m-LEN (STR$ (licz
ba));liczba: RETURN
9030 PRINT AT n,m-LEN (STR$ (lic
zba));liczba: RETURN
```

gdzie *m* jest numerem wybranej kolumny.

• Drukowanie liczby z dwoma miejscami po przecinku z wyrównaniem części całkowitej liczby do kolumny m:

```
9035 REM procedura dotyczy liczb
 z dwoma miejscami po kropce dzi
esietnej
9040 LET b$=(" " AND a$(1)<>"-")
+("-" AND a$(1)="-")
9050 IF b$="-" THEN LET a$=a$(2
TO )
9060 IF LEN a$=1 THEN LET a$=a$+
"0"
9070 IF LEN a$=2 THEN LET a$="0"
+a$
9080 PRINT TAB m-LEN a$;b$;a$( T
O LEN a$-2);".";a$(LEN a$-1 TO )
: RETURN
```

Podana procedura ma zastosowanie do wydruku liczb z przedziału (−999999.99; 999999.995). Przy zmianie liczby cyfr drukowanych po kropce dziesiętnej granice wspomnianego przedziału ulegną nieznacznym zmianom. Ograniczenie to bierze się ze sposobu drukowania liczb przez ZX Spectrum. Działanie powyższej procedury można zademonstrować za pomocą programu:

```
  10 LET a=PI/12345: LET m=10
  20 FOR i=1 TO 10: LET a$=STR$
INT (a*100+.5)
  30 GO SUB 9040: LET a=a*10: NE
XT i
  40 STOP
```

Aby wydruk rozpoczynał się od wybranego na ekranie wiersza, należy linię 9070 w poprzednim podprogramie zmienić na:

```
9070 PRINT AT n,m-LEN a$; a$ (TO LEN a$-2); ".";
     a$ (LEN a$-1 TO): RETURN
```

Drukowanie liczby z wyrównaniem prawostronnym do dowolnie wybranej kolumny można zrealizować znacznie prościej

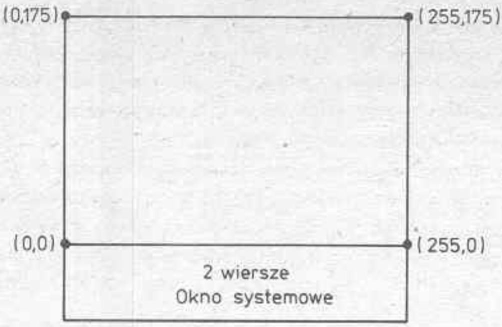
```
9090 PRINT AT n,m-LEN STR$ (INT
(liczba)); INT (liczba*100)/100:
RETURN
9100 PRINT TAB m-LEN STR$ (INT (
liczba)); INT (liczba*100)/100: R
ETURN
```

W tym przypadku wartości gotowe do wydruku nie muszą być zamieniane na łańcuchy znaków. Parametrami powyższej procedury są: numer wiersza oraz numer kolumny, w której ma się kończyć część całkowita drukowanej na ekranie liczby oraz sama liczba z dowolnie ustaloną liczbą miejsc po przecinku (w tym przypadku dwa miejsca po przecinku)*).

Tryb dużej rozdzielczości 3.4.2

Basic ZX Spectrum daje możliwość tworzenia obrazów w trybie dużej rozdzielczości — punkt po punkcie. Obrazy można tworzyć tylko w części ekranu poza oknem systemowym. W trybie dużej rozdzielczości ekran jest podzielony na 176 wierszy po 256 punktów w każdym. Lewy dolny róg ekranu bezpośrednio nad oknem systemowym ma współrzędne x = 0 oraz y = 0. Prawy dolny róg ekranu ma współrzędne (255,0), prawy górny (255,175), a lewy górny (0,175), tak jak na rys. 15. W trybie dużej rozdzielczości współrzędne oznacza się więc

*) Ten uproszczony program nie działa dla liczb z przedziału (−0.01; 0.01).



Rys. 15. Organizacja ekranu w trybie dużej rozdzielczości

odwrotnie niż w trybie znakowym, np. opcji AT y,x czy SCREEN$ (y,x), co jest pewną niekonsekwencją.

Podstawową instrukcją języka Basic umożliwiającą tworzenie rysunków w trybie dużej rozdzielczości jest:

```
PLOT x,y
     ↑↑
     | współrzędna y (0÷175)
     współrzędna x (0÷255)
```

Po wykonaniu PLOT 128,88 zostaje narysowany na ekranie punkt znajdujący się w miejscu określonym argumentami występującymi przy PLOT. Bezpośrednio po włączeniu komputera pozycja PLOT ustawiana jest w punkcie (0,0).

Za pomocą PLOT x,y można rysować wykresy dowolnych funkcji (dowolne krzywe), składając je punkt po punkcie zgodnie z aktualnie obliczonymi współrzędnymi. Narysowanie dowolnej funkcji jest tylko kwestią przyjęcia na ekranie odpowiedniej skali. Prostym tego przykładem może być wykres funkcji kosinus (na razie bez układu współrzędnych):

```
10 FOR x=0 TO 255
20 PLOT x,88+87*COS (x*PI/128)
30 NEXT x
```

Skalowanie zostało przeprowadzone w następujący sposób. Każda pozioma linia na ekranie zawiera 256 punktów (od 0 do 255). Funkcja kosinus jest funkcją okresową o okresie 2π(2*PI). Przyjęto, że okresowi 2*PI odpowiada 256 punktów. Stąd argument funkcji zmienia się w pętli FOR co krotność PI/128. W ten sposób została określona oś odciętych x. Każda pionowa linia na ekranie może zawierać do 176 punktów (od 0 do 175). Funkcja kosinus przyjmuje wartości z przedziału od -1 do 1. Dodanie do współrzędnej liczby podzielonej przez 88 (176:2 = 88) spowoduje umieszczenie wykresu na środku ekranu. Gdyby wartości funkcji kosinus bezpośrednio określały współrzędną y, narysowana zostałaby niemal linia prosta. Aby wykres funkcji był w pełni widoczny na ekranie należy go „rozciągnąć" w pionie. Trzeba zrobić to w taki sposób, aby drugi argument PLOT

określający zmiany wzdłuż osi rzędnych y nigdy nie był mniejszy od zera ani nie większy od 175. Z tego względu w podanym wyżej przykładzie mnoży się wartość funkcji kosinus przez 87, co czyni zadość powyższemu warunkowi. Ostatnim czynnikiem, który należy wziąć pod uwagę przy skalowaniu są proporcje, tzn. omawianą funkcję cos(x) należy tak narysować, aby wykres oddawał w pełni jej przebieg, a nie przypominał przebiegu funkcji cos(2x), czy cos(x/2).

W omawianym przykładzie funkcja kosinus została narysowana w skali liniowej. W wielu przypadkach potrzebne jest narysowanie krzywej w innej skali, np.: logarytmicznej. Zmiany skali dokonuje się przez odpowiednie skalowanie argumentów instrukcji PLOT. Tą metodą w prosty sposób można narysować np. elipsę:

```
  10 LET r=50
  20 FOR i=0 TO 255
  30 PLOT 127+r*SIN (i*PI/128),8
8+(r*COS (i*PI/128))/2
  40 NEXT i
```

Narysowano elipsę o stosunku półosi 2:1.

Wadą rysowania za pomocą PLOT x,y jest to, że tworzenie rysunku trwa bardzo długo. Program rysujący elipsę wykonuje się około 33 s.

Do rysowania odcinków służy kolejna instrukcja języka Basic operująca w trybie wysokiej rozdzielczości:

DRAW x, y
- y: współrzędna y (0÷175)
- x: współrzędna x (0÷255)

Argumentami instrukcji DRAW są współrzędne końca odcinka, który ma być narysowany, liczone względem jego początku. Odcinek zaczyna się natomiast w punkcie, który był ostatnio narysowany. System zapamiętuje sobie współrzędne tego punktu. Po włączeniu komputer przyjmuje, że jest to lewy dolny róg, czyli punkt o współrzędnych (0,0). Początkiem odcinka może być albo punkt (0,0), albo ostatni narysowany za pomocą PLOT, DRAW czy CIRCLE. Ilustracją działania DRAW jest program rysujący zaklejoną kopertę:

```
  10 PLOT 0,0: DRAW 255,0: DRAW
0,175: DRAW -255,0: DRAW 0,-175:
 DRAW 255,175
  20 PLOT 255,0: DRAW -255,175
```

Początek rysowania jest ustalony przez PLOT 0,0. Następnie wykreślone jest pięć odcinków jeden za drugim — „bez odrywania ręki". Przed rysowaniem ostatniego odcinka należy zmienić jego punkt początkowy. Robi się to poleceniem PLOT 255,0 (ustawienie tego punktu w prawym dolnym rogu ekranu). Argumenty DRAW mogą być ujemne w przeciwieństwie do argumentów PLOT.

Ujemny argument przy DRAW oznacza, że rysowanie odcinka będzie odbywało się w lewą stronę*). I tak:

PLOT 0,0 : DRAW 255,175

rysuje odcinek łączący lewy dolny róg ekranu z prawym górnym rogiem, natomiast

PLOT 255,175 : DRAW −255, −175

rysuje odcinek łączący prawy górny róg ekranu z lewym dolnym rogiem. W pierwszym przypadku w trakcie kreślenia odcinka argumenty x i y rosną, a w drugim maleją. Omówione instrukcje PLOT x,y oraz DRAW x,y można w prosty sposób wykorzystać do zobrazowania tzw. ruchów Browna:

```
  10 REM ruchy browna
  20 BORDER 6: PAPER 4: CLS
  30 LET x1=120: LET y1=88
  40 PLOT x1,y1
  50 LET x=10-INT (RND*21): IF x
1+x>=255 OR x1+x<=0 THEN GO TO 5
0
  60 LET x1=x1+x
  70 LET y=7-INT (RND*15): IF y1
+y>=175 OR y1+y<=0 THEN GO TO 70
  80 LET y1=y1+y
  90 DRAW x,y: GO TO 50
```

Na ekranie telewizora rysowany jest tor wyimaginowanej cząstki odbijającej się od przypadkowo spotkanych sąsiadów. Jest to przykład prostego programu symulacyjnego. Program zatrzymuje się po jednoczesnym naciśnięciu CAPS SHIFT i BREAK.

Rysowanie na ekranie telewizora okręgu punkt po punkcie jest czasochłonne. W repertuarze rozkazów języka Basic ZX Spectrum znajduje się instrukcja CIRCLE służąca do kreślenia okręgów:

CIRCLE x, y, r

- r — promień okręgu
- y — współrzędna y środka okręgu
- x — współrzędna x środka okręgu

Używając instrukcji CIRCLE należy pamiętać o następujących zasadach:

$y+r \leqslant 175$ i $y-r \geqslant 0$

$x+r \leqslant 255$ i $x-r \geqslant 0$

Jeżeli któraś z powyższych nierówności nie będzie spełniona, to wykonanie CIRCLE zostanie przerwane, a na ekranie pojawi się komunikat o błędzie: *Integer out of range* — przekroczony zakres dozwolonych wartości parametru.

Działanie polecenia CIRCLE można zademonstrować w następujący sposób:

*) Interpreter oblicza współrzędne końca dodając do współrzędnych ostatnio narysowanego punktu argumenty instrukcji DRAW.

```
10 FOR r=0 TO 81 STEP 9
20 CIRCLE 128,88,r
30 NEXT r
```

Program ten rysuje dziewięć współśrodkowych okręgów. Wykonanie programu trwa około 7,5 s, a więc ponad cztery razy krócej niż kreślenie jednego kręgu za pomocą PLOT, tzn. punkt po punkcie.

Rysowanie każdego okręgu zawsze rozpoczyna się na prawo od zdefiniowanego środka okręgu. Użytkownik nie ma możliwości zdefiniowania początku rysowania. Odcinek równoległy do osi odciętych rozpoczynający się w punkcie, w którym kończy się rysowanie okręgu, i przechodzący przez okręg nie jest średnicą tego okręgu. Przechodzi on o 8 punktów (ekranu) poniżej środka okręgu.

Instrukcja DRAW może także mieć trzy argumenty:

DRAW x, y, d

- d — liczba radianów określająca długość łuku
- x, y — analogicznie jak w dwuparametrowej instrukcji DRAW

W takiej postaci instrukcja DRAW służy do rysowania wycinków okręgów. Po wykonaniu programu:

10 PLOT 128,88 : DRAW 0, 30, PI

na ekranie zostanie narysowany półokrąg. Argument określający długość łuku może być liczbą ujemną. Wówczas rysowanie odcinka okręgu będzie odbywało się zgodnie z ruchem wskazówek zegara (dla argumentów dodatnich przeciwnie):

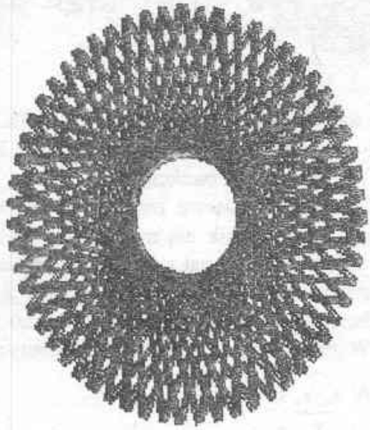
20 PLOT 128,88 : DRAW 0,30, −PI

Po wykonaniu linii 10 i 20 na ekranie telewizora zostanie narysowany okrąg. Jest to już trzeci sposób kreślenia okręgu.

Prosty obraz robiący wrażenie trójwymiarowości jaki można utworzyć z pomocą DRAW prezentuje następujący program:

```
  10 LET n=36: LET a=0  LET d=2*
PI/n: CIRCLE 128,88,80
  20 FOR i=1 TO n
  30 LET x=40*COS a: LET y=40*SI
N a
  40 PLOT 128,88: DRAW x,y,-PI:
DRAW x,y,PI
  50 LET a=a+d
  60 NEXT i
```

Instrukcja DRAW x,y,d pozwala na rysowanie nie tylko wycinków okręgów — mimo że jest to jej podstawowe zadanie. Przykłady tworzenia ciekawych rysunków demonstruje program:



```
10 FOR i=51 TO 254 STEP 4: CLS
20 PLOT 60,40: DRAW 128,88,PI*
(2*i-1)
30 NEXT i
```

W tym przypadku trzeci argument DRAW powinien być nieparzystą wielokrotnością PI. W przeciwnym razie mogą wystąpić błędy w działaniu programu, o czym system operacyjny powiadomi użytkownika odpowiednim komunikatem. Można się o tym przekonać zmieniając linię 20 na:

20 PLOT 60,40: DRAW 128,88, PI*i

Często do tworzenia własnych obrazów na ekranie używa się programów graficznych napisanych przez profesjonalne firmy zajmujące się tworzeniem oprogramowania. Najpopularniejsze z nich to: *Melbourne Draw*, *Leonardo*, *VU-3D*, *Pencil Designers*. Programy takie mają duże możliwości i są pomocne przy projektowaniu grafiki. Podano obok prosty program, który może pomóc przy rysowaniu na ekranie. Można go nazwać „Zaczarowany ołówek 2".
Program ten umożliwia wykreślanie linii w obrębie wydzielonej części ekranu. W linii 30 są wprowadzane współrzędne początkowe x,y, czyli współrzędne punktu, od którego ma się rozpocząć rysowanie. Rysowanie odbywa się punkt po punkcie, za pomocą PLOT — linia 40. Aktualne współrzędne PLOT są obliczane w liniach 50 i 60 na podstawie odczytu stanu klawiszy 1,2 oraz 9,0 (patrz p. 1.6). Naciśnięcie klawisza z cyfrą 1 powoduje rysowanie odcinka na lewo, a klawisza z cyfrą 2 na prawo od wybranego punktu. Rysowanie w górę i w dół jest możliwe za pomocą klawiszy 0 i 9, odpowiednio. Jednoczesne naciśnięcie dwóch klawiszy pozwala na kreślenie w kierunku wypadkowym, np.: naciśnięcie 1 (w lewo) i 0 (góra) spowoduje narysowanie odcinka w kierunku pośrednim. W tym przy-

```
 10 REM zaczarowany olowek 2
 20 LET k=1: POKE 23658,8
 30 INPUT "podaj poczatkowe x=?
,y=?";x;"  ";y
 40 PLOT x,y
 50 LET x=x+((0 OR IN 63486=190
) AND x<255)-((0 OR IN 63486=189
) AND x>0)
 60 LET y=y+((0 OR IN 61438=190
) AND y<175)-((0 OR IN 61438=189
) AND y>0)
 70 PRINT AT 21,18;"x=";x;"  ";
"y=";y;"  "
 80 IF INKEY$="D" THEN LET k=1
 90 IF INKEY$="G" THEN LET k=2
100 GO TO 30+K*10
```

padku będzie to kierunek lewo—góra lub używając nomenklatury geograficznej kierunek północno-zachodni. W programie tym używa się dodatkowo dwóch klawiszy sterujących „D" i „G". Klawisz „D" oznacza — opuść „pióro". Natomiast „G" oznacza — podnieś pióro. Podczas gdy „pióro jest podniesione" poruszając się nie zostawia za sobą śladu na ekranie. Informacje o miejscu, w którym aktualnie znajduje się „pióro", można odczytać na dole ekranu jako wartości współrzędnych x i y.

Druga instrukcja w linii 20 (POKE 23658,8) powoduje ustawienie trybu pracy klawiatury na duże litery — tryb „C". Adres występujący przy instrukcji POKE przyporządkowany jest do zmiennej systemowej FLAGS2 (dodatek C). Jedynka na trzecim bicie tej zmiennej ustawia tryb dużych liter. Dzięki temu w liniach 80 i 90 użyto bez obawy liter „D" i „G" (wykonując instrukcję POKE 23658,0 ustawia się tryb L pracy klawiatury tzw. tryb małych liter).

Jeżeli 40 linie programu uzupełni się do postaci:

30 PLOT OVER 1; x,y

to wówczas powtórne rysowanie po wcześniej określonych liniach będzie powodowało ich wytarcie.

W instrukcjach PLOT i DRAW można użyć opcji PAPER, INK, BRIGHT, FLASH, OVER oraz INVERSE. Wstawia się je bezpośrednio po PLOT czy DRAW i oddziela się od siebie lub argumentów średnikami.

Po tym wyjaśnieniu zaprezentowany program graficzny nie ma już chyba dla Czytelników żadnych tajemnic. Dlatego też jako zadanie do samodzielnego rozwiązania proponuje się udoskonalenie tego programu przez wprowadzenie dodatkowych możliwości, np.: widoczne oznaczenie aktualnego położenia „pióra".

Z instrukcji języka Basic używanych w trybie dużej rozdzielczości do omówienia została funkcja POINT. Ma ona dwa argumenty, które muszą być współrzędnymi punktu znajdującego się na ekranie. Funkcja ta przyjmuje dwie wartości: zero — gdy punkt o określonych argumentami współrzędnych ma kolor tła, lub 1 — gdy punkt ten ma kolor atramentu.

Przesuwanie na ekranie obrazów zrealizowanych w trybie wysokiej rozdzielczości jest dosyć złożone. Prostą metodę przesuwania zawartości ekranu przedstawiono w poniższym przykładzie. Podany program rysuje na ekranie w płaszczyźnie pionowej sinusoidę. Co osiem narysowanych linii, na skutek działania instrukcji PRINT "", wykres jest przesuwany o osiem linii (tzn. o jeden wiersz). Po wypełnieniu całego ekranu program w zależności od odpowiedzi na pytanie: „scroll?" kontynuuje wydruk, bądź kończy działanie.

```
  10 LET l=0: PRINT AT 21,0;"":
REM poczatkowa pozycja wydruku
  20 POKE 23692,3: LET k=INT 127
*SIN (PI*l/64)
  30 LET l=l+1
  40 IF l-INT (l/8)*8=0 THEN PRI
NT ""
  50 PLOT 127,7-(l-INT (l/8)*8):
 DRAW k,0
  60 GO TO 20
```

* Ekran 3.5

Pamięć obrazu zajmuje 6144 bajty od adresu 16384 do 22527. Każdemu punktowi na ekranie odpowiada jeden bit. Cały ekran (kontrolowany przez komputer) składa się z 49152 punktów (pixeli), wobec tego pamięć ekranu zajmuje 49152/8 = = 6144 bajty.

Podczas wczytywania danych do pamięci obrazu ZX Spectrum można się przekonać jak pamięć ta jest zorganizowana. Ilustruje to program zaczerniający punkt po punkcie cały ekran:

```
10 FOR i=16384 TO 22527
20 POKE i,255
30 NEXT i
40 PAUSE 0
```

W czasie wykonywania programu widać, że punkty w jednej linii (o szerokości jednego punktu) są zapalane jeden za drugim. Jednakże poszczególne linie ekranu nie są zapalane jedna za drugą!

Ekran jest podzielony na trzy części — tak jak na rys. 16, z których każda składa się z 64 linii po 256 punktów. Na początku wczytywania danych do pamięci obrazu wprowadzane są dane do linii zerowej (pierwszej części — górnej 1/3 ekranu). Następnie zapalane są punkty w liniach 8, 16, ... 56, a po tym zapalane są punkty w liniach 1, 9, 17, ... 57 itd. Procedura ta powtarza się do momentu zapalenia wszystkich punktów górnej 1/3 ekranu. W analogiczny sposób są wczytywane dane do środkowej, a następnie dolnej części ekranu.

Gdyby w programie nie było linii 40, to po zaczarnianiu ekranu dwie dolne linie — okno systemowe — przyjęłoby kolor brzegu ekranu (BORDER) i w przed-



Rys. 16. Podział ekranu

ostatniej linii ekranu ukazałby się komunikat OK : 30,1, linie te bowiem powtórnie wykorzystywane są przez system operacyjny.

Można powiedzieć, że każda z trzech części, na które podzielony jest ekran w trybie znakowym składa się z 8 wierszy po 32 znaki. Adres, od którego rozpoczyna się wpisywanie danego znaku w danej części ekranu, można w prosty sposób obliczyć. Dla wierszy od 0 do 7 (górna część ekranu) oblicza się go według wzoru:

$$\text{Adres} = 16384 + Y*32 + X$$

gdzie:

Y — numer wiersza, tu od $0 \div 7$,
X — numer kolumny od $0 \div 31$,

dla wierszy od 8 do 15 (środkowa część ekranu):

$$\text{Adres} = 16384 + 2048 + (Y-8)*32 + X$$

gdzie:

Y — numer wiersza, tu od $8 \div 15$,
X — numer kolumny od $0 \div 31$,

dla wierszy 16–23 (dolna część ekranu)

$$\text{Adres} = 16384 + 4096 + (Y-16)*32 + X$$

gdzie:

Y — numer wiersza, tu od $16 \div 23$,
X — numer kolumny od $0 \div 31$.

Wzory te można zastąpić jednym wyrażeniem:

$$\text{Adres} = 16384 + 2048*\text{INT}(Y/8) + 32*Y - \text{INT}(Y/8)*8*32 + X$$

Przy takiej organizacji ekranu do sterowania wczytaniem dowolnego znaku lub całego obrazu (ekranu) ZX Spectrum potrzebuje dwóch bajtów pamięci RAM. Są one przyporządkowane zmiennej systemowej o nazwie DF CC o adresie 23684. Funkcje każdego z bitów tej zmiennej są przedstawione na rys. 17.

Analogiczne, jednak znacznie bardziej skomplikowane wyrażenie służące

Numer bitu															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Wartość															
0	1	0													
Adres początku obszaru pamięci obrazu równy 16384			Bity b12 i b11 są równe 0 zapełniana jest górna 1/3 ekranu; b12=0 b11=1 zapełniana jest środkowa 1/3 ekranu; b12=1 i b11=1 zapełniona jest dolna 1/3 ekranu		Bity b10, b9 i b8 wskazują systemowi, który bajt danego znaku ma zostać aktualnie wpisany do pamięci: b10=b9=b8=0 pierwszy bajt (0) b10=b9=b8=1 ostatni bajt (7)			Bity b7, b6 i b5 wskazują systemowi, który wiersz (w odpowiedniej 1/3 ekranu) mają być wpisane kolejne znaki. Wiersze są numerowane od 0 do 7			Dzięki tym bitom zlokalizowana jest aktualna współrzędna X danego znaku na ekranie				

Rys. 17. Organizacja zmiennej systemowej DF CC

do obliczania adresu (w pamięci obrazu) dowolnego punktu o współrzędnych X,Y, można napisać w następujący sposób (wyznaczamy adres bajtu):

$$
\begin{aligned}
\text{Adres } (X,Y) = {} & 16384+32*(\text{INT}((175-Y)/8)-\text{INT}((175-Y)/\\
& /64)*8+8*(175-Y-\text{INT}((175-Y)/8)*8+64*\\
& *\text{INT}((175-Y)/64))+\text{INT}(X/8)
\end{aligned}
$$

Informacje dotyczące numeru bitu w danym bajcie oblicza się w następujący sposób:

$$\text{nr bitu} = 7-x+\text{INT}(X/7)*7$$

(bity numerowane są od $0 \div 7$).

Znając organizację pamięci ekranu można tworzyć obrazy operując bezpośrednio na jej pamięci. Do tego celu można użyć instrukcji:

POKE *adres w pamięci ekranu*, *liczba*
↑
liczba z zakresu od 0 do 255

Taka metoda tworzenia obrazów jest bardzo pracochłonna i mało efektywna. W praktyce można z niej skorzystać tylko do „kosmetyki" projektowanej grafiki. Zapełnienie obrazu punkt po punkcie można także zrobić na poziomie języka asemblera.

Po wpisaniu do pamięci obrazu kształtów zaprojektowanych grafik wczytywane są atrybuty. Atrybuty wczytywane są kolejno począwszy od lewego górnego rogu do prawego dolnego rogu ekranu, kolumna za kolumną wiersz za wierszem — nie tak jak dane do pamięci obrazu (dokładne omówienie w p. 3.1 i 3.3).

Obszar pamięci od adresu 65368 do 65535 jest przeznaczony na tzw. grafiki definiowane przez użytkownika UDG (ang. *User Defined Graphics*). UDG pozwala zdefiniować 21 dowolnych, dodatkowych znaków. Adres początku obszaru UDG jest zapamiętany w zmiennej systemowej UDG — bajty o adresach 23675 (mniej znaczący bajt) i 23676 (bardziej znaczący bajt). Można więc samemu zmienić adres początku obszaru grafik definiowanych nadając zmiennej UDG nową wartość:

POKE 23675,*l*
↑
mniej znaczący bajt nowej wartości adresu UDG

POKE 23676,*h*
↑
bardziej znaczący bajt nowej wartości adresu UDG

Należy jednak pamiętać, że nowy adres początku UDG powinien być umieszczony w obszarze pamięci niedostępnym dla Basica, czyli powyżej adresu ostatniego bajtu dostępnego dla Basica, pamiętanego w zmiennej systemowej RAMTOP o adresie 23730 i 23731.

ZX Spectrum na grafiki definiowane standardowo rezerwuje 168 bajtów pamięci. Ponieważ każdy znak jest jednoznacznie określony przez osiem bajtów (pole 8 × 8 punktów) w obszarze UDG można zdefiniować 21 własnych symboli graficznych. Po załączeniu zasilania system operacyjny ZX Spectrum wpisuje do obszaru grafik definiowanych kształty dużych liter od A do U.

Często zachodzi potrzeba skorzystania ze znaku, którego jest brak w zestawie znaków danego mikrokomputera. Przykładem tego mogą być polskie litery jak: ę, ą, ć, ś, lub litery greckie ρ, Ω, Σ itd. wykorzystywane do oznaczania konkretnych wielkości lub operacji w naukach ścisłych. Podstawową informacją potrzebną do zdefiniowania nowego znaku jest to, że każdemu bitowi w bajcie przyporządkowana jest wartość zero lub jeden — zgodnie z reprezentacją binarną. Poza tym każdy bit ma swoją wagę odpowiadającą potędze liczby dwa (rys. 18). Zamianę liczby z reprezentacji binarnej (zero-jedynkowej) na dziesiętną dla pojedynczego bajtu realizuje funkcja:

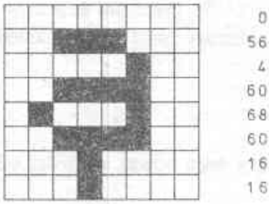
$$F(\text{bajt}) = 128*f(\text{bit7})+64*f(\text{bit6})+32*f(\text{bit5})+16*f(\text{bit4})+ \\ +8*f(\text{bit3})+4*f(\text{bit2})+2*f(\text{bit1})+f(\text{bit0})$$

gdzie: f(bit i) przyjmuje wartość zero lub jeden w zależności od wartości danego

Bajt								
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
$2^7=128$	$2^6=64$	$2^5=32$	$2^4=16$	$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$	Waga bitu

Rys. 18. Znaczenie bitów bajtu pamięci

bitu. Z wiadomości wstępnych o grafice wynika, że jedynka w określonym bicie powoduje zapalenie odpowiadającego mu punktu na ekranie po wprowadzeniu zawartości danego bajtu do obszaru pamięci obrazu. Natomiast zero powoduje zgaszenie danego punktu. Można więc przystąpić do projektowania własnych grafik definiowanych np. litery ą.



Rys. 19. Przykład grafiki definiowanej (litera ą)

Po zrobieniu projektu na papierze (rys. 19) należy odpowiednie wartości umieścić w pamięci mikrokomputera choćby pod adresem odpowiadającym pierwszemu znakowi UDG, czyli literze A. Zapis danych do pamięci mikrokomputera realizuje jeden z programów (do wyboru a lub b)

a

```
10 FOR i=0 TO 7
20 READ n: POKE USR "a"+i,n
30 NEXT i
40 DATA 0,0,56,4,68,60,0
```

b

```
10 FOR i=0 TO 7
20 INPUT "wczytaj";(i);" bajt
 UDG";n
30 POKE USR "A"+i,n
40 NEXT i
```

Funkcja:

USR "*litera*"

daje w wyniku adres, w obszarze UDG, znaku ujętego w cudzysłowy. Argumentami USR mogą być litery od A do U włącznie lub od a do u, co ma to samo znaczenie.

W programach a) i b) można dokonać, w zależności od potrzeb zmiany sposobu wczytania wartości *n*. W obu przypadkach *n* jest liczbą w reprezentacji dziesiętnej, możliwe jest wczytanie *n* w reprezentacji binarnej. Do tego celu używa się funkcji BIN. Liczb binarnych można używać zamiast dziesiętnych, np. wpisanie do pamięci liczby 109 wyglądałoby następująco:

POKE USR "A" + i BIN 01101101

jest to równoważne:

POKE USR "A" + i, 109

Funkcja BIN dokonuje konwersji swojego argumentu, z postaci binarnej na odpowiadającą mu liczbę dziesiętną, np.: po wykonaniu polecenia: PRINT BIN 11111111 równoważnego PRINT 255 na ekranie zostanie napisana liczba 255.

Jak już wspomniano za pomocą grafik definiowanych przez użytkownika można rozszerzyć gamę znaków ZX Spectrum choćby o polskie litery. Rozszerzenie to dla małych polskich liter realizuje poniższa procedura:

```
9850 LET a$="acelnostu": REM lit
ery w trybie G ,ktore zostana za
stapione polskimi malymi literam
i
9860 FOR i=1 TO LEN a$: FOR j=0
TO 7
9870 READ n: POKE USR a$(i TO i)
+j,n
9880 NEXT j: NEXT i
9890 DATA 0,56,4,60,68,60,16,16:
 REM litera A w trybie G
9900 DATA 4,8,28,32,32,32,28,0:
REM litera C wtrybie G
9910 DATA 0,56,68,120,64,60,8,8:
 REM litera E w trybie G
9920 DATA 0,16,24,16,48,16,12,0:
 REM litera L w trybie G
9930 DATA 8,16,120,68,68,68,68,0
: REM litera L w trybie G
9940 DATA 8,16,56,68,68,68,56,0:
 REM litera O w trybie G
9950 DATA 8,16,56,64,56,4,120,0:
 REM litera S w trybie G
9960 DATA 16,0,124,8,16,32,124,0
: REM litera T w trybie G
9970 DATA 8,16,124,8,16,32,124,0
: REM litera U w trybie G
9980 PRINT a$: PRINT "ąćęłńóśżź"
9990 RETURN
```

Po wykonaniu tej procedury, z klawiatury będą dostępne bezpośrednio w trybie graficznym polskie litery takie jak: ą, ć, ę, ł, ń, ó, ś, ż, ź. Są one wpisane w obszarze UDG zamiast liter podanych w łańcuchu z linii 9850 i dostępne w trybie graficznym pracy klawiatury. Procedura ta może zostać wywołana, jeżeli w programie wykona się polecenie GO SUB 9850.

Zdefiniowanie małych polskich liter zajęło 72 bity z obszaru grafik definiowanych. W obszarze UDG jest jeszcze wystarczająca ilość pamięci, aby wpisać tam dane odpowiadające określeniu kształtu dużych polskich liter. Niech będzie to zadaniem dla Czytelnika. Bardzo pomocny podczas rozwiązywania postawionego zadania będzie program służący do projektowania UDG, który znajduje się na kasecie demonstracyjnej z zestawu ZX Spectrum. Program ten w wersji oryginalnej ma tytuł: *Character Generator*, a jedną z jego opcji jest opcja UDG.

Jako że obszar pamięci przeznaczony na UDG znajduje się ponad obszarem dostępnym dla Basica, to po wykonaniu, służącej do definiowania UDG procedury, można ją usunąć z pamięci mikrokomputera wykonując instrukcję:

NEW. Nowo zdefiniowane znaki nie zostaną usunięte z pamięci komputera, gdyż NEW zeruje pamięć RAM tylko w obszarze dostępnym dla Basica. Chcąc wielokrotnie korzystać z wprowadzonych tą drogą rozszerzeń należy zmieniony przez użytkownika obszar grafik definiowanych zapisać na taśmę za pomocą polecenia (dla ZX Spectrum 48 k)

SAVE "pol. litery" CODE 65368,168

lub (dla ZX Spectrum 16 k)

SAVE "pol. litery" CODE 32640,168

Jeżeli kiedykolwiek zajdzie potrzeba pisania programu z wykorzystaniem polskich liter, to przed rozpoczęciem pracy z komputerem trzeba wczytać do obszaru UDG zdefiniowany wyżej zbiór znaków (np. LOAD""CODE). Można także zdefiniować i nagrać na taśmę wiele innych zbiorów znaków w zależności od potrzeb. W każdym ze zbiorów można określać dowolne znaki pod warunkiem, że nie będzie ich nigdy więcej niż 21.

Po takiej prezentacji grafik definiowanych Czytelnik mógłby przypuszczać, że UDG używa się przede wszystkim do rozszerzenia liczby znaków (liter osiąganych bezpośrednio z klawiatury). Tak nie jest. Grafiki definiowane wykorzystuje się głównie do projektowania (rysowania) z elementów o wielkości znaku dowolnych obrazów na ekranie. Przykład wykorzystania tego narzędzia jakim dysponuje programista ZX Spectrum demonstruje program:

```
  10 BORDER 5: PAPER 5: CLS : GO
SUB 1000
  20 PLOT 60,71: DRAW 135,0
  30 PAUSE 50
  40 PRINT AT 10,14;"AB";AT 10,1
8;"  "
  50 PRINT AT 11,14;"CD";AT 11,1
8;"AB"
  60 PRINT AT 12,14;"  ";AT 11,1
8;"EF"
  70 PAUSE 50
  80 PRINT AT 10,14;"  ";AT 10,1
8;"AB"
  90 PRINT AT 11,14;"AB";AT 11,1
8;"CD"
 100 PRINT AT 12,14;"EF";AT 12,1
8;"  "
 110 GO TO 30
1000 REM definiowanie grafik uzy
tkownika
1010 FOR i=0 TO 47
1020 READ n: POKE USR "a"+i,n
1030 NEXT i
1040 DATA 0,2,3,7,15,19,83,126,0
,64,64,224,240,200,202,126,127,1
15,120,31,15,4,4,28,254,206,60,2
48,240,32,56,0,127,127,120,27,15
,4,28,0,254,254,30,216,240,32,32
,56
1050 RETURN
```

Znaki drukowane instrukcjami PRINT wprowadzono w trybie G. Po wykonaniu podprogramu (od linii 1000) przyjmą one postać jak na wydruku:

```
  10 BORDER 5: PAPER 5: CLS : GO
SUB 1000
  20 PLOT 60,71: DRAW 135,0
  30 PAUSE 50
  40 PRINT AT 10,14;"";AT 10,1
8;" "
  50 PRINT AT 11,14;"";AT 11,1
8;""
  60 PRINT AT 12,14;" ";AT 12,1
8;""
  70 PAUSE 50
  80 PRINT AT 10,14;" ";AT 10,1
8;""
  90 PRINT AT 11,14;"";AT 11,1
8;""
 100 PRINT AT 12,14;"";AT 12,1
8;" "
 110 GO TO 30
1000 REM definiowanie grafik uzy
tkownika
1010 FOR i=0 TO 47
1020 READ n: POKE USR "a"+i,n
1030 NEXT i
1040 DATA 0,2,3,7,15,19,83,126,0
,64,64,224,240,200,202,126,127,1
15,120,31,15,4,4,28,254,206,30,2
48,240,32,56,0,127,127,120,27,15
,4,28,0,254,254,30,216,240,32,32
,56
1050 RETURN
```

Program ten wprowadza animację (ruchome rysunki) realizowaną za pomocą instrukcji PRINT oraz znaków zdefiniowanych przez użytkownika w obszarze UDG. Naciśnięcie dowolnego klawisza przez okres dłuższy od 0,7 s przyspiesza ruch UFO-ludków. Ruch takich figur sterowany z poziomu Basica jest jednakże dość powolny, w związku z tym w programach profesjonalnych do ich sterowania używa się procedur napisanych w kodzie maszynowym. Znaki, których aktualne współrzędne na ekranie i wygląd są zmieniane przez wpisanie danych do pewnych miejsc pamięci i wywołanie podprogramu animacji nazywa się (z ang.) sprite'ami. Niektóre komputery domowe mają sprzętowy mechanizm realizacji sprite'ów, np. Commodore 64, Atari 800 XL. W ZX Spectrum należy to robić programowo. W punkcie 7.2 opisano wersję języka Basic dla ZX Spectrum (Mega-Basic) z możliwością tworzenia sprite'ów.

* * Generator znaków 3.7

W ostatnich 768 bajtach pamięci stałej ROM ZX Spectrum znajduje się **g e n e r a t o r z n a k ó w** (ang. *Character Generator*). Umieszczone są tam kształty

znaków należących do zbioru ASCII (ang. *American Standart Code of Information Interchange*). Adres początku generatora znaków zawiera 2-bajtowa zmienna systemowa o nazwie CHAR$ i adresie 23606 i 23607. Po wykonaniu w trybie bezpośrednim (natychmiastowym) instrukcji:

PRINT PEEK 23606+256*PEEK 23607

na ekranie zostanie napisana liczba 15360. Jest ona zmniejszonym o 256 adresem generatora znaków. W rzeczywistości adres generatora znaków zawartego w ROM-ie jest równy 15616.

W ZX Spectrum znaki ASCII mają kody od 32 do 127. Znak spacji (odstępu) ma kod 32, natomiast znak © — ostatni z generatora znaków — kod 127 (patrz dodatek A). Wszystkie znaki oraz polecenia języka Basic wyświetlane na ekranie (oprócz znaków graficznych) są tworzone na podstawie zdefiniowanych w pamięci stałej znaków. Znaki używane w Spectrum oraz ich kody można wydrukować za pomocą prostego programu:

```
10 FOR i=32 TO 127
20 PRINT i,CHR$ i
30 NEXT i
```

Program ten drukuje w lewej kolumnie kod, a w prawej odpowiadający mu znak. Jednakże jest on mało poglądowy. Bardziej interesujące byłoby wydrukowanie struktury każdego ze zdefiniowanych w pamięci ROM znaków, bajt po bajcie.

```
  10 FOR i=15616 TO 16383
  20 POKE 23692,3
  30 LET a=PEEK i
  40 FOR j=0 TO 7
  50 PRINT AT 21,15-j,a-2*INT (a
/2)
  60 LET a=INT (a/2)
  70 NEXT j
  80 PRINT AT 21,0;i;AT 21,17;PE
EK i''
  90 IF i=15616 THEN GO TO 110
 100 IF (i-15616)/8=INT ((i-1561
6)/8) THEN PRINT : PAUSE 0
 110 NEXT i
 120 NEXT i
```

Najważniejszą częścią programu są linie od 40 do 70. Tu liczba dziesiętna otrzymana za pomocą funkcji PEEK jest zamieniana na sekwencję zerojedynkową ilustrującą, które punkty znaku na ekranie będą zapalone (jedynki), a które zgaszone (zera). Taka postać każdej komórki pamięci o adresie *i* jest drukowana w 22 linii ekranu, tuż nad oknem systemowym. W tej linii drukowany jest adres odpowiedniej komórki oraz jej zawartość w reprezentacji dziesiętnej. Realizuje to 80 linia programu, która jest zakończona dwoma apostrofami (nie mylić z cu-

dzysłowem). Apostrofy te są równoważne zmianie pozycji wydruku o dwie linie w dół. Jeżeli w programie nie byłoby linii 20, to po wydrukowaniu zawartości komórki o adresie 15616 komputer zapytałby „scroll?". Dalsze wydruki nastąpiłyby po naciśnięciu dowolnego klawisza (prócz BREAK oraz N lub SPACE w Spectrum +). Aby tego uniknąć do zmiennej systemowej SCR CT o adresie 23692 odpowiadającej za liczbę wierszy drukowanych na ekranie bez przewijania (scroll), została wpisana liczba 3 (w pętli ze zmienną sterującą *i*). Czytelność wydruku poprawiają linie 90 i 100. Dzięki umieszczeniu ich w programie, po wydrukowaniu zawartości ośmiu kolejnych bajtów odpowiadających zdefiniowaniu jednego znaku jest drukowana linia pusta separująca kolejne znaki. Po wydrukowaniu ośmiu kolejnych bajtów program zatrzymuje się do momentu naciśnięcia dowolnego klawisza. Pozwala to na zapoznanie się ze strukturą danego znaku bez pośpiechu.

Powiększone obrazy znaków zdefiniowanych w generatorze znaków będą bardziej czytelne, jeżeli w wydruku w reprezentacji binarnej wszystkie zera zastąpi się spacjami. W tym celu linię 50 należy zastąpić następującym fragmentem programu.

```
45 IF a-2*INT (a/2)=0 THEN LET
a$=" ": GO TO 55
50 LET a$="1"
55 PRINT AT 21,15-j;a$
```

Wydruki z poprzedniego programu oraz jego zmodyfikowanej wersji będą wyglądały tak jak na rys. 20.

Omówione do tej pory programy dotyczące generatora znaków drukują na ekranie zawartość ostatnich 768 bajtów ROM-u — bajt po bajcie. Nie ma

a

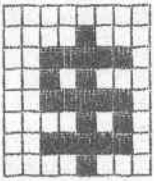
15976	00000000	0
15977	01000010	66
15978	01100110	102
15979	01011010	90
15980	01000010	66
15981	01000010	66
15982	01000010	66
15983	00000000	0

b

15976									0
15977		1					1		66
15978		1	1			1	1		102
15979		1		1	1		1		90
15980		1					1		66
15981		1					1		66
15982		1					1		66
15983									0

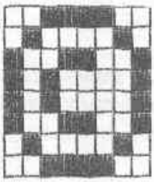
Rys. 20. Generator znaków:
a — zawartość poszczególnych bajtów wybranego znaku, b — wyświetlenie „zapalonych" punktów w danym bajcie

w nich możliwości obejrzenia struktury dowolnie wybranego znaku, bez oglądania pozostałych. W sposób poglądowy można zrealizować to:



```
0      15648
8      15649
62     15650
40     15651    $
62     15652
10     15653
62     15654
8      15655
```

```
76543210
nr bitu zawar.  adres  znak
```



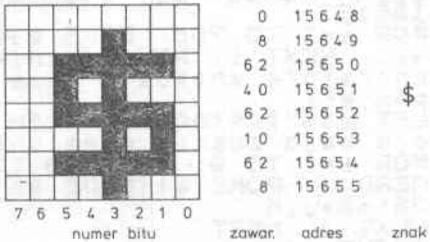
```
60     16376
66     16377
153    16378
161    16379    ©
161    16380
153    16381
66     16382
60     16383
```

```
76543210
nr bitu zawar.  adres  znak
```

```
   5 REM program ROM ACSII
  10 POKE 23658,8: REM wlaczenie
 trybu duzych liter
  15 REM dane wejsciowe
  20 CLS : INPUT "wczytaj dowoln
y pojedynczy znak";a$
  25 IF CODE a$<32 OR CODE a$>12
7 THEN GO TO 20
  30 LET adres=15616+(CODE a$-32
)*8: LET z=0
  35 REM interpretacja bajtu o a
dresie i
  40 FOR i=adres TO adres+7: LET
 k=PEEK i
  45 FOR j=7 TO 0 STEP -1
  50 PRINT AT z,j;"█" AND (k-2*I
NT (k/2))=1
  55 LET k=INT (k/2)
  60 NEXT j
  65 PRINT AT z,9;PEEK i;AT z,16
;i: LET z=z+1
  70 NEXT i
  75 REM uzupelniajacy opis wydr
uku
  80 PRINT AT 9,0;"76543210"
  85 PRINT AT 10,0;"nr bitu zawa
r.  adres  znak"
```

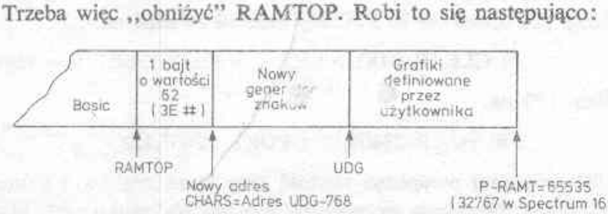
```
 90 PRINT AT 3,24; FLASH 1;a$
 95 REM ramka
 100 FOR i=0 TO 8: PLOT i*8,111:
 DRAW 0,64: NEXT i
 105 FOR i=0 TO 8: PLOT 0,i*8+11
1: DRAW 64,0: NEXT i
 110 REM czykontynuowac program
 115 INPUT "czy wczytac kolejny
znak"; LINE c$
 120 IF c$<>"T" AND c$<>"N" THEN
 GO TO 115
 125 IF c$="T" THEN GO TO 20
 130 STOP
```

Poszczególne części programu zostały opatrzone odpowiednimi komentarzami Programy opatrzone komentarzami są czytelne i łatwe do zrozumienia nawet po upływie pewnego czasu od jego napisania. Przykładowy wydruk z tego programu przedstawiono na rys. 21.



Rys. 21. Ilustracja zawartości generatora znaków — przykładowy wydruk

Można zdefiniować własny generator znaków. Robi się to kosztem zmniejszenia dostępnej pamięci RAM. Adres początku generatora znaków jest zapamiętany w dwubajtowej zmiennej systemowej CHARS o adresie 23606 (mniej znaczący bajt adresu) oraz 23607 (bardziej znaczący bajt adresu). Po załączeniu zasilania system operacyjny ZX Spectrum wpisuje pod te adresy wartości odpowiednio 0 i 60. Własny generator znaków można umieścić w pamięci RAM np. zaraz za obszarem grafik definiowanych (rys. 22). Przed dokonaniem tej operacji należy miejsce na nowy generator znaków zabezpieczyć przed zniszczeniem przez Basic. Trzeba więc „obniżyć" RAMTOP. Robi to się następująco:



Rys. 22. Lokalizacja nowego generatora znaków w pamięci ZX Spectrum

CLEAR PEEK 23675+256*PEEK 23676-769

gdzie komórki 23675 i 23674 zawierają adres obszaru UDG dla ZX Spectrum 48 k: CLEAR 64599, a dla ZX Spectrum 16 k CLEAR 31831.

Na rysunku 22 pokazano, że pamięć RAM dostępna dla Basica jest oddzielona od pozostałej części pamięci RAM bajtem o wartości 62 (heksadecymalnie 3E). Z tego względu od nowego adresu RAMTOP należy dodatkowo odjąć 1.

Zdefiniowany teraz zostanie nowy generator znaków tak by miał on pełny polski alfabet w zakresie małych liter. Przed przystąpieniem do realizacji tego zadania należy — na podstawie dotychczasowych doświadczeń — usunąć te znaki ze zbioru znaków ZX Spectrum, których prawie się nie używa podczas pisania własnych programów. Mogą to być następujące znaki: !,@, %, §, -, £, &. Wybranym znakom przyporządkowuje się teraz polskie małe litery np.:

! — ą;@ — ć; % — ę; & — ł ;_ — ń; £ — ó; | — ś; ~ — ż; / — ź

Dalszą część pracy wykona program:

```
  10 CLEAR 64599: LET a=64600: L
ET b=15615
  20 FOR i=1 TO 768: POKE a+i,PE
EK (b+i): NEXT i: REM przeniesie
nie generatora znakow w nowe mie
jsce pamieci
  30 LET a$="ąćęłńóśżź": REM zna
ki ktore maja zostac podmienione
  40 FOR i=1 TO 9: FOR j=0 TO 7
  50 READ n: POKE a+(CODE a$(i T
O i)-32)*8+j,n
  60 NEXT j: NEXT
  70 PRINT a$
  80 DATA 0,56,4,60,68,60,16,16
  90 DATA 4,8,28,32,32,32,28,0
 100 DATA 0,56,68,120,64,60,8,8
 110 DATA 0,16,24,16,48,16,12,0
 120 DATA 8,16,120,68,68,68,68,0
 130 DATA 8,16,56,68,68,68,56,0
 140 DATA 8,16,56,64,56,4,120,0
 150 DATA 16,0,124,8,16,32,124,0
 160 DATA 8,16,124,8,16,32,124,0
 170 POKE 23606,89: POKE 23607,2
51
 180 PRINT a$
 190 STOP
```

Dla wersji ZX Spectrum 16 k należy linię 10 zmienić na:

10 CLEAR 31831: LET a = 31832 : LET b = 15615

oraz linię 170 na:

170 POKE 23606,88 : POKE 23607,123

Linia 20 programu przepisuje wartość generatora znaków z pamięci ROM do pamięci RAM. Następnie wczytywany jest łańcuch znaków a$, którego elementami są „niepotrzebne" znaki mające być zastąpione polskimi małymi literami.

Program od linii 40 do 60 podmienia znaki łańcucha a$ zgodnie z danymi występującymi po słowach kluczowych DATA (80÷160). Nowy adres generatora znaków ustawia 170 linia programu. Po wczytaniu w trybie bezpośrednim instrukcji:

PRINT PEEK 23606+256*PEEK 23607

na ekranie zostanie wydrukowana liczba 64344 (dla wersji 16 k — 31576). Jest to nowy adres generatora znaków zmniejszony o 256 (zgodnie z wcześniejszą uwagą). Po wydrukowaniu się na ekranie łańcucha a$ ze zmodyfikowanymi znakami można nowy generator znaków zapamiętać na taśmie za pomocą:

SAVE "polznak" CODE 64600,768 (dla wersji 48 k)

lub

SAVE "polznak" CODE 31832,768 (dla wersji 16 k)

oraz zweryfikować dokonany zapis:

VERIFY "polznak" CODE

Program „polznak" może być dołączony do pisanych przez użytkowników ZX Spectrum programów, w których występują komentarze lub opisy obiektów wykorzystujące polskie litery. Należy jednak pamiętać, że przed każdorazowym użyciem programu „polznak" trzeba zmienić adres generatora znaków z 15360 na 64344 (lub 31832).

W podmienionym generatorze znaków nie ma dużych polskich liter Ć, Ł, Ó, Ś, Ż, Ź, od których mogą rozpoczynać się polskie wyrazy. Stosując konsekwentnie przedstawioną wyżej metodę należy uzupełnić łańcuch a$ dodatkowymi sześcioma znakami. Zmienić górne ograniczenie pętli w linii 40 ze zmienną sterującą *i* z 9 na 15 oraz uzupełnić listę danych informacjami o kształcie kolejnych sześciu znaków.

Programy komputerowe podczas działania operują niekiedy na kilkunastu lub więcej obrazach zajmujących cały ekran. Na wygenerowanie każdego takiego obrazu ZX Spectrum potrzebuje 6912 bajtów. W pamięci RAM (49152 bajty) mieści się tylko około 7 takich obrazów. Jedną z metod rozwiązania tej trudności jest podmiana generatora znaków znajdującego się w ROM-ie oraz składanie obrazów z fragmentów o wielkości znaku każdy. Zdefiniowane tak kształty znaków podmienia się kształtami fragmentów obrazów, z których później tworzy się gafiki na ekranie telewizora. Obraz tworzy się za pomocą instrukcji PRINT. Dzieje się to podobnie jak w przypadku wykorzystania grafik definiowanych przez użytkownika z tym, że generator znaków daje większe możliwości. A mianowicie:
— można w nim zdefiniować nie 21, lecz 96 własnych znaków graficznych,
— słowa kluczowe języka Basic składane są z dużych liter będących elementami generatora znaków (kody od 65 do 90). Zawartość nowego generatora znaków może być tak dobrana, że po wydrukowaniu wybranego słowa kluczowego na ekranie ukaże się fragment rysunku.

Drugą z możliwości ilustruje program, w którym dla uproszczenia cały generator znaków wypełniono kształtem tylko jednego znaku:

```
  10 CLEAR 64599: LET a=64600
  20 FOR i=1 TO 768 STEP 8: FOR
j=0 TO 7
  30 READ n: POKE a+i+j,n
  40 NEXT j
  50 RESTORE 130
  60 NEXT i
  70 CLS : INPUT "wczytaj liczbe
 od 166 do 255 ";n: IF n<166 OR
n>255 THEN GO TO 80
  80 PRINT "stary gen.znak  nowy
 gen.znak."
  90 PRINT AT 1,7;CHR$ n: POKE 2
3606,89: POKE 23607,251
 100 PRINT AT 1,22;CHR$ n: POKE
23606,0: POKE 23607,60
 110 PRINT #0;"nacisnij dowolny
klawisz": PAUSE 0: GO TO 70
 120 STOP
 130 DATA 146,146,255,146,146,25
5,146,146
```

Zaletą korzystania ze zmienionego generatora znaków czy też grafik definiowanych przez użytkownika podczas tworzenia obrazów na ekranie telewizora jest to, że program, za pomocą którego otrzymuje się na ekranie żądany obraz, zajmuje dużo mniej miejsca w pamięci niż program robiący to samo punkt po punkcie. Jest to więc mniej pracochłonna metoda tworzenia grafiki.

Części statyczne i dynamiczne jednego obrazu na ekranie telewizora mogą powstawać nawet opierając się na kilku różnych generatorach znaków. Wymiana bieżącego zbioru znaków na inny nie spowoduje zamiany kształtów na ekranie. Dzieje się tak dlatego, że obraz ZX Spectrum jest generowany na podstawie obszaru pamięci obrazu. Raz wpisane do tego obrazu pamięci kształty trwają w nim do momentu wpisania na ich miejsce nowych. Nie są one zależne od aktualnego generatora znaków.

* Współpraca z drukarką 3.8

Na zakończenie warto jeszcze wspomnieć o możliwości wyprowadzania wyników działania programu na drukarkę działającą bez pośrednictwa dodatkowego układu sprzęgającego (np. ZX PRINTER, SEIKOSHA GP 50 A). Instrukcja LPRINT pozwala wyprowadzić informację na drukarkę w identyczny sposób jak instrukcja PRINT wyprowadza ją na ekran. Ze zrozumiałych względów nie wszystkie zlecenia dopuszczalne w PRINT mają swoje odpowiedniki w instrukcji LPRINT.

Wydruk realizowany za pomocą LPRINT może być tabulowany dzięki użyciu opcji TAB, wysuwany do odpowiedniej kolumny przez AT (numer wiersza jest tutaj ignorowany); niektóre kody znaków sterujących, np. kod BRIGHT, prze-

kazywane funkcją CHR$ także będą działać. INVERSE i OVER mają w przypadku LPRINT niezmienione znaczenie; podobnie separatory: „,", „;", „'" (przecinek, średnik, apostrof). Użycie innych opcji jest zabronione.

Obraz powstały na ekranie, także w trybie wysokiej rozdzielczości, można skopiować na drukarkę instrukcją COPY. Ta bezparametrowa dyrektywa powoduje wydrukowanie punkt po punkcie zawartości 176 linii ekranu (22 wiersze), przy czym wszystkie punkty o kolorze atramentu są drukowane, a punkty w kolorze tła nie są (pozostaje kolor papieru).

Treść programu w języku Basic może być wydrukowana na drukarce za pomocą instrukcji LLIST.

Twórcy układów dopasowujących inne drukarki do ZX Spectrum wprowadzają niekiedy zmiany w realizacji powyższych instrukcji. Przed użyciem takich układów autorzy doradzają dokładne zapoznanie się z ich możliwościami i sposobem obsługi.

** Między Basicem a asemblerem 4

* * Wprowadzenie do asemblera 4.1

W rozdziale 1 napisano, że procesor Z80 jest „mózgiem" Spectrum. Niestety jest to tylko metafora. Procesor nie myśli. Wykonuje natomiast programy oraz steruje pracą innych elementów komputera. Procesor jest automatem. Automat ten potrafi wykonywać kilkaset ustalonych (zaprogramowanych z góry) operacji. W Z80 jest ich ponad 500. Operacje te nazwano instrukcjami maszynowymi lub instrukcjami języka wewnętrznego. Operacje wykonywane przez procesor to proste czynności na binarnych liczbach zapisanych na jednym lub dwu bajtach. Liczby pobierane są z pamięci lub rejestrów procesora. Rejestry są układami elektronicznymi zawartymi wewnątrz procesora, które mogą pamiętać bajt lub dwubajtowe słowo.

Każda instrukcja maszynowa ma własny kod. W Z80 kod zajmuje jeden lub dwa bajty. Informuje on procesor jakie czynności należy wykonać — innymi słowy steruje jego pracą.

Niektóre instrukcje maszynowe oprócz kodu zawierają dodatkowe informacje, np. adres komórki pamięci, z której należy pobrać dane. Te dodatkowe informacje łącznie z kodem instrukcji wskazują sposób uzyskania argumentów instrukcji, czyli liczb, na których należy wykonać daną operację.

Program w języku maszynowym jest ciągiem instrukcji, kodów wraz z argumentami, umieszczonych po kolei w pamięci. Procesor automatycznie pobiera kolejne instrukcje i wykonuje je. Dokładniejsze informacje o pracy procesora znaleźć można w p. 4.2.

Układanie programu w języku wewnętrznym za pomocą kodów instrukcji i adresów jest bardzo niewygodne. Dlatego też opracowano specjalne języki przeznaczone do pisania takich programów. W językach tych instrukcje maszynowe mają nazwy — tzw. mnemoniki. Są one zazwyczaj skrótami słów angielskich określających funkcję danej instrukcji, np. ADD z ang. *add*—dodaj LD z ang. *load*— załaduj, itd.

Języki takie określają także sposób zapisywania argumentów instrukcji,

liczb, nazw zmiennych itd. Nazwano je językami niskiego rzędu, niskiego poziomu lub językami asemblerowymi*). Programy, które tłumaczą program zapisany w takim języku na binarny program maszynowy nazywają się Asemblerami (w p. 7.10 opisano wybrany Asembler przeznaczony dla ZX Spectrum).

Pisanie, a zwłaszcza uruchamianie i testowanie programów w języku asemblera jest bardzo żmudne i pracochłonne. Oto niedostatki programowania w tym języku (w gwarze informatycznej mówi się: programowania w asemblerze):

- w języku asemblera trudniej pisać, analizować i uruchamiać programy niż w języku wyższego rzędu, np. Pascalu czy Basicu,
- bardzo trudno pisać w asemblerze programy arytmetyczne oraz programy współpracujące z urządzeniami we/wy,
- programów (napisanych w języku asemblera) nie można uruchomić na innym procesorze, np. programów napisanych dla Z80 nie da się z reguły przetransponować na inny procesor,
- program asemblerowy składa się ze znacznie większej liczby instrukcji niż równoważny funkcjonalnie program w języku wyższego rzędu.

Mimo znacznych niedostatków programowanie w języku asemblera daje ogromne korzyści. Główne zalety to:

- szybkość działania; w ZX Spectrum programy napisane w języku asemblera wykonują się co najmniej 60 razy szybciej niż program w Basicu,
- bardziej efektywne użycie pamięci (zmiennych),
- program w asemblerze zajmuje w pamięci mniej miejsca niż program (równoważny) napisany w języku wyższego rzędu**),
- programy takie umożliwiają bezpośredni dostęp do sprzętu i urządzeń zewnętrznych, pozwalają wykorzystać wszystkie możliwości procesora oraz działają niezależnie od systemu operacyjnego; w przypadku użycia języków wyższego rzędu nie zawsze jest to możliwe.

Największą zaletą programów napisanych w języku asemblera jest szybkość działania. Niektóre programy np. system operacyjny, a także programy graficzne, generujące dźwięki lub wykonujące wiele obliczeń (np. przy analizie sygnałów) powinny być napisane w języku niskiego poziomu.

* * Sposób pracy procesora. Programowanie w języku asemblera 4.2

Informacje podane w tym oraz następnym punkcie są zbyt skąpe, by nauczyć programowania w asemblerze. Stanowią jednak wstęp, który może pomóc w zro-

*) W książce przyjęto pisownię spolszczoną słowa *assembler* jako asembler, termin ten bowiem na trwałe znalazł się w słowniku informatycznym.

**) Zawiera więcej instrukcji niż równoważny program napisany w języku wyższego rzędu, lecz każda instrukcja języka wyższego rzędu zamieniana jest na kilkanaście instrukcji maszynowych, podczas gdy każdej instrukcji asemblera odpowiada tylko jedna instrukcja maszynowa.

zumieniu zasad użycia tego języka. Wiadomości można uzupełnić na podstawie książek [6, 20, 21].

Procesor wykonuje ciąg instrukcji (kodów operacji wraz z argumentami) w ten sposób, że automatycznie pobiera z pamięci pojedynczy bajt. ZX Spectrum jest tak zbudowany, że włączenie zasilania powoduje, iż jako pierwszy zostanie pobrany bajt zapisany w komórce o adresie 0. Specjalny rejestr zawarty wewnątrz procesora tzw. licznik rozkazów (oznaczany często PC od ang. *Program Counter*) wskazuje, z którego miejsca pamięci należy pobrać bajt zawierający kod operacji. Procesor dekoduje go. Pobiera, jeśli potrzeba, dodatkowe bajty - argumenty operacji, wykonuje odpowiednie czynności oraz zwiększa zawartość licznika rozkazów, który wskazuje teraz na następną instrukcję do wykonania. Zazwyczaj zwiększa jego zawartość o tyle, ile bajtów wynosiła długość wykonywanej instrukcji. W Z80 instrukcja zajmuje od 1 do 4 bajtów pamięci. Jeśli operacja jest instrukcją skoku, to do licznika rozkazów wpisuje się wyliczony w czasie jej wykonywania adres skoku. Następnie procesor wykonuje kolejną instrukcję, itd.

Podczas wykonywania instrukcji procesor generuje dodatkowo wiele sygnałów, które pojawiają się na różnych wyjściach procesora w różnych fazach wykonywania instrukcji np. sygnały zapis czy odczyt (RD, WR), sygnał uaktywniający układy pamięci (MREQ) lub układy wejścia/wyjścia (IORQ) itd. Są one potrzebne po to, by procesor mógł współpracować z innymi układami komputera. Dla programisty są one na ogół zupełnie nieistotne.

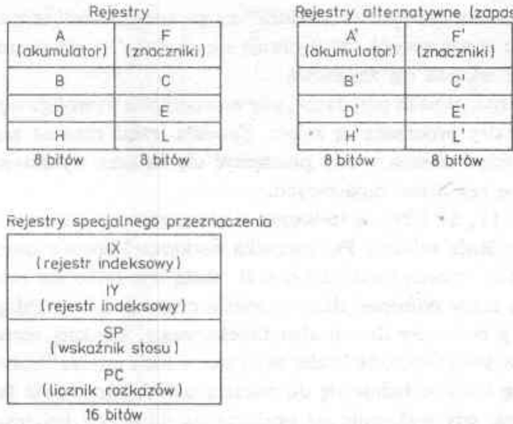
Dla osoby programującej w języku asemblera ważna jest architektura rejestrów procesora (ich liczba i organizacja) oraz lista instrukcji, czyli spis wszystkich instrukcji maszynowych, które procesor potrafi wykonywać.

Architektura rejestrów procesora

Wewnątrz procesora Z80 znajduje się kilkanaście rejestrów jedno- lub dwubajtowych. W rejestrach przechowuje się aktualnie obliczane wartości oraz adresy pamięci. Można sobie wyobrazić rejestry na podobieństwo zmiennych, takich jak w języku Basic, przy czym liczba tych zmiennych jest ograniczona. Niekiedy zawartość rejestru musi być zapisana w pamięci, gdyż nie wystarcza ich do przekazywania wszystkich potrzebnych danych. Dostęp do nich zabiera bardzo mało czasu, znacznie mniej niż odczyt z pamięci. Dlatego w rejestrach umieszcza się aktualnie obliczane wartości. Rejestry procesora Z80 pokazano na rys. 23.

Procesor ma 7 rejestrów 8-bitowych: A, B, C, D, E, H, L oraz tzw. rejestr znaczników F. Sześć rejestrów, tzn. rejestry oznaczone literami B, C, D, E, H, L, można używać także jako rejestry 16-bitowe. 16 bitów zapamiętane jest wtedy w parze rejestrów: BC, DE lub HL (pierwszy rejestr z pary jest bardziej znaczący — B, D, H).

Literą A oznaczono akumulator (nazwa jest pozostałością z dawnych czasów, gdy komputery miały tylko jeden rejestr przeznaczony do wykonywania



Rys. 23. Rejestry procesora Z80

operacji). Jest to wyróżniony, najbardziej wygodny rejestr. Większość operacji arytmetycznych i logicznych można w Z80 wykonywać tylko na informacjach zapisanych w tym rejestrze.

Rejestr F to rejestr znaczników. Nie jest to rejestr, ale zbiór ośmiu jednobitowych rejestrów, z których tylko sześć jest używanych. W czasie wykonywania niektórych instrukcji zapisuje się w nich wartość 1 lub 0. Najczęściej używa się znacznika przeniesienia — C (ang. *carry*)[*]. Informuje on czy nastąpił nadmiar w czasie wykonywania operacji arytmetycznej. Jeżeli dodaje się dwie liczby ośmiobitowe, to niekiedy wynik zajmuje dziewięć bitów i nie mieści się w rejestrze, to wtedy C jest tym dodatkowym bitem. Podobnie rzecz się ma, gdy dodajemy liczby 16-bitowe. Gdy wykonywana jest operacja odejmowania, to bit C informuje, czy odejmowano większą liczbę od mniejszej.

Ważnym znacznikiem jest także Z (zero). Sygnalizuje on czy po wykonaniu operacji w rejestrze znajduje się liczba zero (wszystkie bity mają wartość zero). Znacznika Z używa się najczęściej przy organizowaniu pętli w programie.

Często korzysta się ze znacznika S (z ang. *sign* — znak). Mówi on, czy wynik operacji jest liczbą dodatnią czy ujemną. Wskaźnik S jest ustawiany na wartość 1, jeśli wynik (po pewnych operacjach 16-bitowy, w innych 8-bitowy) na najbardziej znaczącym bicie ma 1, natomiast na 0, kiedy najbardziej znaczący bit jest równy 0[**].

Procesor Z80 ma także grupę rejestrów zapasowych (alternatywnych). Wykonanie pojedynczej instrukcji maszynowej pozwala odłączyć rejestry A, F, B, C, D, E, H, L i w ich miejsce podłączyć: A', F', C', D', E', H', L'. Ta sama

[*] Oznaczamy czasem Cy.

[**] Liczby binarne traktowane są niekiedy w trochę inny sposób niż wyjaśniony (patrz p. 3.6). Ów inny sposób interpretacji liczb nosi nazwę kodu U2 (kod ten został opisany w dalszej części rozdziału).

instrukcja wykonana raz jeszcze ,,włącza" na powrót ,,stare" rejestry. Można to sobie wyobrazić w ten sposób, że wyjmuje się ,,kostkę" z rejestrami z procesora i na jej miejsce wkłada się zapasową.

Rejestry alternatywne są przydatne, gdy w programie wywołuje się podprogram, a wszystkie rejestry procesora są zajęte. Zamiast tracić czas na zapamiętywanie rejestrów w pamięci można wtedy przełączyć się na czas wykonywania podprogramu na grupę rejestrów zapasowych.

Rejestry IX, IY, SP i PC są 16-bitowe. Są one głównie używane do zapamiętywania adresu. Rola rejestru PC (licznika rozkazów) została omówiona. Rejestry IX, IY (tzw. rejestry indeksowe) służą nie tylko do przechowywania adresów. Mogą także pamiętać dane — wśród operacji Z80 znajdują się rozkazy dodawania pary rejestrów do rejestru indeksowego. Częściej jednak używa się tych rejestrów w ten sposób, że liczba zapisana w nich jest traktowana jako adres pamięci, według którego ładuje się do rejestru bajt*). Możliwość taka jest szczególnie przydatna, gdy wykonuje się operacje na sekwencji kolejnych liczb zapisanych w pamięci. Można używać wtedy pętli — podobnie jak w języku Basic. Adres aktualnie przetwarzanego elementu jest umieszczany w rejestrze IX lub IY, a po każdorazowym wykonaniu pętli zwiększa się go. Cechą charakterystyczną operacji wykorzystujących rejestry indeksowe jest możliwość adresowania bloku 256 bajtów względem ich aktualnej zawartości.

Wskaźnik stosu SP (z ang. *Stack Pointer*) służy do adresowania pamięci w pewien wyróżniony sposób. Część pamięci komputera jest zorganizowana jako stos (podobnie jak np. stos książek, łatwo można sięgnąć po element leżący na szczycie; element włożony na stos ostatnio zostanie zdjęty jako pierwszy itd.). Procesor wykonuje operacje zapisu na stos (PUSH) i odczytu ze stosu (POP) dwu bajtów, uaktualniając odpowiednio zawartość rejestru SP. Na stosie zapamiętywane są także adresy powrotu w czasie wywołania podprogramu.

Uwaga *Stos rozbudowuje się w dół pamięci, tzn. ku mniejszym adresom. W ZX Spectrum stos jest umieszczony na ,,szczycie" pamięci.*

Lista rozkazów

Operacje, które może wykonywać procesor są bardzo proste. Dla wykonania jednej operacji Basica trzeba czasem wykonać kilkanaście (kilkadziesiąt) instrukcji maszynowych. Można je podzielić na kilka grup:
— przesyłanie bajtu z pamięci do rejestru, z rejestru do rejestru lub rejestru do pamięci,
— przesłanie słowa, tzn. pary bajtów (jak wyżej),
— operacje arytmetyczne (tylko dodawanie i odejmowanie) słów lub bajtów,
— przesłanie bajtu z/do urządzenia wejścia—wyjścia,

*) Bardzo często do adresowania pamięci używa się pary rejestrów HL, niekiedy także BC i DE.

— operacje logiczne (iloczyn logiczny, negacja, porównywanie, przesuwanie — tylko bajtów (ustawianie i testowanie poszczególnych bitów),
— operacja przetwarzania bloku operacji, np. przepisywanie bloku z jednego miejsca pamięci w drugie,
— operacje sterujące, tj. takie, które zmieniają zawartość licznika rozkazów (skok, wywołanie podprogramu, powrót z podprogramu, przerwanie programowe i powrót z przerwania),
— operacje działające na stosie,
— inne operacje, które trudno sklasyfikować, np.: nic nie rób, ustawienie wskaźników procesora itd.

W zbiorze instrukcji procesora można wyróżnić grupy po kilka instrukcji, które wykonują te same czynności, a różnią się jedynie sposobem uzyskania argumentów instrukcji. Sposób w jaki podaje się argumenty lub ich adresy nazywa się trybem adresowania. W procesorze Z80 używa się następujących trybów adresowania:
— rejestrowy — operacja jest wykonywana na zawartości rejestrów, np.: zapisz w rejestrze A zawartość rejestru B — LD A, B,
— natychmiastowy — w instrukcji maszynowej jest umieszczana liczba, która jest argumentem operacji np.: zapisz w rejestrze A liczbę 100 — LD A,100,
— bezpośredni — w instrukcji maszynowej jest umieszczany adres liczby, która ma być argumentem operacji, np. załaduj do rejestru A liczbę zapisaną w miejscu pamięci o adresie 100; LD A,(100),
— indeksowy — adres otrzymuje się dodając liczbę umieszczoną w instrukcji do zawartości jednego z rejestrów indeksowych; IX lub IY, np.: LD A,(IX+10) — załaduj do akumulatora zawartość komórki o adresie równym zawartości rejestru IX zwiększonego o 10,
— pośredni — zawartość pary rejestrów stanowi adres, np. LD A,(HL) — załaduj do akumulatora zawartość komórki wskazywaną przez parę HL,
— implikowany — adres argumentu wynika z kodu operacji, np. CCF — zaneguj znacznik C,
— względny (tylko w instrukcji skoku względnego) — adres skoku wylicza się dodając do aktualnej zawartości licznika rozkazów (w czasie wykonania) liczbę podaną w instrukcji. W Z80 istnieje także instrukcja skoku bezwzględnego, której argumentem jest adres skoku,
— adresowanie względem zawartości wskaźnika stosu — w czasie wykonywania operacji nie oblicza się adresu, lecz informacje pobiera się lub zapisuje ze szczytu stosu, np. załaduj do rejestru A oraz rejestrów znaczników dwa bajty ze szczytu stosu i zmniejsz stos — POP AF.

Uwaga *Procesor Z80 nie pozwala na to, by każda operacja mogła przetwarzać bajt uzyskany wg dowolnego trybu adresowania. Pewne operacje mogą wykorzystywać tylko niektóre tryby.*

Na początku tego rozdziału stwierdzono, że w języku asemblera trudno jest zaprogramować obsługę urządzenia zewnętrznego oraz obliczenia arytmetyczne.

Wynika to z faktu, że procesor nie potrafi wykonywać operacji we/wy oprócz przesłania pojedynczego bajtu lub grupy bajtów. Dlatego należy pisać podprogramy, które taką współpracę realizują.

Podobnie rzecz się ma z arytmetyką. Repertuar rozkazów arytmetycznych Z80 jest bardzo ubogi. Procesor może wykonywać tylko dodawanie, także dodawanie 1, tzw. inkrementacja, i odejmowanie, także zmniejszanie o 1 — tzw. dekrementacja. Samemu należy napisać podprogramy innych operacji arytmetycznych, np. mnożenie, dzielenie itd.

Jak wiadomo, na komputerze każda liczba reprezentowana jest ciągiem bitów — zer lub jedynek. Ciąg ten może być interpretowany na różne sposoby. Najczęściej używa się tzw. naturalnego kodu binarnego (NKB). Kod ten był już opisywany, jednak dla przypomnienia: w kodzie tym wartość liczby oblicza się w następujący sposób (przykład dla liczby jednobajtowej):

$$N \cdot 2^7 + N \cdot 2^6 + \ldots + N \cdot 2^0$$

gdzie N — 1 lub 0 oznacza wartość danego bitu (kolejno siódmego, szóstego, ... aż do zerowego). Podobnie dla liczby szesnastobitowej:

$$N \cdot 2^{15} + N \cdot 2^{14} + \ldots + N \cdot 2^0$$

gdzie N — 1 lub 0 są wartościami kolejnych bitów: najpierw bajtu bardziej znaczącego, następnie mniej znaczącego. Na pojedynczym bajcie można zapisywać liczby z zakresu $0 \div 255$ tzn. $0 \div 2^8 - 1$. Na dwu bajtach liczby z zakresu $0 \div 65535$ tzn. $0 \div 2^{16} - 1$. Często używa się także kodu uzupełnieniowego do 2 (U2). Pozwala on interpretować liczbę jako ujemną bądź dodatnią. Liczbę w kodzie U2 oblicza się następująco:

$$N \cdot (-2)^7 + N \cdot 2^6 + \ldots + N \cdot 2^0$$

↑

oto różnica między NKB i U2

Na pojedynczym bajcie można zapisać liczby z zakresu od —128 do 127 ($-2^7 \div 2^7 - 1$), a na dwu bajtach liczby z zakresu — $2^{15} \div 2^{15} - 1$.

Uwaga *Jeżeli wykonuje się operacje maszynowe: dodawanie lub odejmowanie, to niezależnie od tego czy interpretujemy argumenty jako zakodowane w NKB czy w U2 wynik będzie odpowiednio poprawny w obu kodach (znacznik C należy potraktować jako najbardziej znaczący bit wyniku).*

W książce tej ze względu na zakres tematu nie opisywano dokładnie działania wszystkich instrukcji maszynowych procesora Z80. Ograniczono się jedynie do umieszczenia w dodatku B spisu wszystkich rozkazów procesora. W dodatku tym podano następujące informacje:
— mnemoniki instrukcji,
— dozwolone w danej instrukcji tryby adresowania,
— czynność, jaką dany rozkaz wykonuje,
— liczby bajtów pamięci zajmowanych przez dany rozkaz,
— czas wykonania (w taktach zegara — jeden takt trwa ok. 0,000 0003 s),

— wpływ wykonania instrukcji na stan znaczników procesora.
Kody binarne instrukcji podano w dodatku A.

Procesor może obsługiwać także tzw. p r z e r w a n i a (ang. *interrupt.*) Podanie sygnału na odpowiednie wejście procesora: INT lub NMI, czy wykonanie specjalnej instrukcji — RST, powoduje, że procesor przerywa aktualnie wykonywany program i zaczyna wykonywać zapisany w ustalonym miejscu pamięci podprogram obsługi przerwań.

Procesor Z80 może przyjmować przerwania maskowane w trzech trybach oraz niemaskowane. Bliższe informacje o przerwaniach w Z80 znaleźć można w książce [20]*).

System ZX Spectrum jest tak zorganizowany, że użytkownik może wykorzystać jedynie przerwanie maskowane w trybie 2. Wymaga to zarówno odpowiedniego oprogramowania, jak i specjalnego układu dołączonego do Spectrum.

* * Język asemblera ZX Spectrum 4.3

Język asemblera jest specjalnym językiem przeznaczonym do zapisywania programów maszynowych. Jedna instrukcja asemblera odpowiada jednej instrukcji maszynowej. Poznanie zasad zapisywania takich rozkazów pomoże w zrozumieniu listy instrukcji procesora zawartej w dodatku B.

Instrukcja asemblera zajmuje jedną linię programu. Forma tej linii jest następująca:

```
Start   LD    HL,1     ; pierwsza linia
 ↑      ↑      ↑         ↑
etykieta|      |         |
        mnemonik         |
        (nazwa)|         |
        instrukcji       |
               argumenty |
               (jeden lub dwa)
                         komentarz
                         (rozpoczyna się znakiem ;)
```

Etykiety wykorzystuje się m.in. w instrukcjach skoku. W asemblerze linie nie są numerowane tak jak w Basicu. Funkcję, która w instrukcji GO TO *nr*, GO SUB *nr* spełnia numer linii, w asemblerze pełni etykieta, która jest nazwą rozpoczynającą się od litery. Może zawierać litery i cyfry. Etykietami należy opatrzeć tylko te linie, do których występują skoki, lub w których zaczyna się podprogram. Pozostałe linie nie muszą zawierać etykiety. Mnemonik pełni funkcję słowa klu-

*) Zrezygnowano z bliższego opisu przerwań, bowiem w ZX Spectrum użytkownik ma niewielkie możliwości ich wykorzystania. Ograniczono się więc do udzielenia wskazówki dla Czytelników obeznanych z techniką przerwań.

czowego. Mnemoniki instrukcji procesora Z80 są standaryzowane. Zawsze używa się tych samych mnemoników — takich, jakie podano w dodatku B.

Argumenty*) informują o tym skąd należy pobrać informację lub gdzie należy ją przesłać.

Wśród instrukcji Z80 występują instrukcje:
— bezargumentowe, np. CCF,
— jednoargumentowe, np. POP BC,
— dwuargumentowe, np. LD IX,2000.

Uwaga *W instrukcjach dwuargumentowych pierwszy argument określa, gdzie przesłać informację, drugi podaje źródło informacji.*

Argumenty informują o trybie adresowania użytym w danej instrukcji. Mogą one mieć następującą postać:
— nazwa rejestru lub pary rejestrów oznacza tryb rejestrowy,
— nazwa pary rejestrów lub rejestru indeksowego ujęta w nawiasy oznacza odpowiednio adresowanie pośrednie lub indeksowe,
— liczba oznacza adresowanie natychmiastowe,
— liczba ujęta w nawiasy oznacza adresowanie bezpośrednie.

Tryb adresowania względem zawartości wskaźnika stosu oraz tryb implikowany wynikają z mnemonika instrukcji.

Średnik ustawiony w odpowiednim miejscu linii pełni funkcję podobną, jak słowo kluczowe REM w języku Basic. Wszystkie znaki umieszczone po średniku są komentarzem. Komentarza nie musimy używać w instrukcjach.

W językach niskiego poziomu oprócz instrukcji występują także p s e u d o - i n s t r u k c j e, tj. takie linie programu, które nie są zastępowane binarną instrukcją maszynową. Pozwalają one m.in. na zadeklarowanie zmiennej: bajtu, słowa lub całego bloku pamięci. Zmiennej takiej nadaje się nazwę. W programie nazwy zmiennej można używać jako argumentu instrukcji zamiast adresu zmiennej.

Pseudoinstrukcje określają także od jakiego miejsca pamięci należy umieścić kod maszynowy, definiują stałe, itd. Opis wybranego programu, tzw. asemblera, który tłumaczy program napisany w języku niskiego rzędu na binarny kod oraz dodatkowe informacje o języku znaleźć można w p. 8.10.

* * Jak korzystać z języka asemblera ZX Spectrum? 4.4

Pewne operacje muszą być oprogramowane w asemblerze. Język Basic ZX Spectrum ma specjalną funkcję USR, która pozwala łączyć program napisany w języku niskiego poziomu z programem w Basicu. Z Basicu można wywołać program

*) Argumenty dwubajtowe umieszczane są w pamięci komputera w ten sposób, że najpierw jest bajt mniej znaczący, potem bardziej.

napisany w języku maszynowym zakończony instrukcją RET. Po jego zakończeniu sterowanie wraca do Basica.

USR jest funkcją i dlatego nie można jej użyć samodzielnie w linii programu, lecz musi wystąpić np. w instrukcji PRINT USR *adr*, GOTO USR *adr*, LET zmienna = USR *adr* itd. Najczęściej używa się instrukcji:

RANDOMIZE USR *adr*

↑ adres procedury asemblerowej umieszczonej w pamięci komputera

Wywołanie funkcji USR może być umieszczone w programie. Może także być wykonane w trybie bezpośrednim. Za pomocą tej funkcji można wywoływać podprogramy zawarte w systemie operacyjnym ZX Spectrum. Ciekawe przykłady użycia procedur systemowych opisano w p. 4.5.

Po zakończeniu działania podprogramu wywołanego przez USR, zawartość pary rejestrów BC jest traktowana jako wynik jej działania. Nie zawsze jednak wynik jest potrzebny. W takim przypadku wywołanie funkcji może mieć postać: RANDOMIZE USR *adr*. Przypadkowa zawartość pary rejestrów BC służy wtedy wyłącznie do „przestawienia" generatora liczb losowych — jest z punktu widzenia programisty ignorowana.

Jednak czasem pożądane jest, by procedura asemblerowa obliczała jakieś wartości. Gdy procedura taka zostanie napisana w ten sposób, że w parze BC zostanie umieszczony wynik obliczeń, to po wywołaniu postaci:

LET A = USR *adr*

w zmiennej A znajdzie się wynik obliczony w procedurze asemblerowej.

W jakim miejscu pamięci umieścić można program binarny?

Program taki powinien być umieszczony w pamięci tak, by program w Basicu nie zniszczył go. Najlepszą metodą jest umieszczenie go w obszarze niedostępnym dla Basicu, a więc powyżej komórki wskazywanej przez zmienną systemową RAMTOP (23730, 23731). Zmienna ta wskazuje na ostatni bajt dostępny dla interpretera języka Basic.

Jeżeli wartość zmiennej RAMTOP zmniejszy się (w gwarze informatycznej mówi się — „obniży się" RAMTOP), to zarezerwowany zostanie pewien obszar pamięci RAM. Nową wartość zmiennej RAMTOP można wpisać za pomocą instrukcji CLEAR *nowy adres*. Należy pamiętać, że CLEAR ustawia wartość RAMTOP oraz zawartość zmiennych systemowych używanych przez interpreter Basica i kasuje zmienne programu w Basicu. Program umieszczony powyżej RAMTOP jest zabezpieczony przed zniszczeniem i może być wywołany za pomocą USR.

Program binarny można umieścić w buforze drukarki, tzn. w komórkach

o adresach od 23296 do 23552 (256 bajtów). Będzie on zabezpieczony przed zniszczeniem, gdy program w Basicu nie korzysta z drukarki*).

Można także zarezerwować pamięć w obszarze definicji kanałów. Problem ten został dokładnie opisany w p. 4.6. Umieszczenie programu w tym obszarze może jednak prowadzić do błędu, choć prawdopodobieństwo błędu jest bardzo małe. Jeżeli w programie napisanym w języku Basic nastąpi próba otwarcia niezdefiniowanego kanału (patrz p. 4.6) i gdy przypadkiem w określonych miejscach programu binarnego wystąpią bajty, które mogłyby zostać uznane przez interpreter za nazwę takiego błędnego kanału, to system może się zawiesić.

Program binarny może zostać zapamiętany wewnątrz programu napisanego w języku Basic. Pamięć rezerwuje się wtedy za pomocą instrukcji REM. REM powinien wystąpić jako pierwsza instrukcja programu. Po słowie kluczowym REM należy napisać tyle dowolnych znaków, np.: spacji, ile bajtów pamięci potrzeba na zapamiętanie programu binarnego.

Adres początku programu binarnego jest równy zawartości zmiennej systemowej PROG+5 (patrz rozdz. 2.5). W Basicu można go wyznaczyć:

```
LET adres = PEEK (23636)*256+PEEK(23635)+5
```

Jak umieścić w pamięci program binarny?

Program taki można wczytać z magnetofonu do pamięci komputera za pomocą instrukcji LOAD""CODE (patrz p. 1.5). Instrukcja ta może być wykonana w trybie bezpośrednim lub zapisana w programie w Basicu. Wykonanie instrukcji LOAD""CODE nie niszczy programu w Basicu (o ile nie ingeruje się w obszar programu i zmiennych systemowych).

Program binarny można także umieścić w zbiorze DATA programu napisanego w języku Basic w postaci ciągu liczb. Program zapamiętany w zbiorze DATA jest odczytywany za pomocą instrukcji READ, a poszczególne bajty są umieszczane w pamięci za pomocą POKE. Metoda ta jest najczęściej stosowana w programach publikowanych w czasopismach. Jest także używana w tej książce. Przykład jej użycia znajduje się m.in. w rozdz. 5 i w dalszej części rozdziału.

Jak przekazywać parametry do programu napisanego w języku asemblera?

Niekiedy należy przekazać do podprogramu asemblerowego pewne informacje — parametry, podobnie jak ma to miejsce w Basicu (patrz p. 2.2.6). Parametry można przekazać za pomocą wybranych komórek pamięci.

Istnieje także bardziej wyrafinowana metoda. Wywołanie procedury asemblerowej należy umieścić w instrukcji DEF FN np.:

*) Tego obszaru pamięci używa np. ZX Printer lub Seikosha GP 50 A. Zeruje go także instrukcja COPY.

10 DEF FN a(x,y) = USR *adres*

nazwa funkcji

parametry, które mają być przekazane do procedury asemblerowej (w tym przykładzie funkcja ma dwa parametry)

Przy takim wywołaniu procedury zawartość pary rejestrów BC jest zwracana jako wynik funkcji *a*.

Przekazanie parametrów n i e   n a s t ę p u j e   a u t o m a t y c z n i e. Po wywołaniu funkcji zdefiniowanej jak wyżej, np.:

20 PRINT FN a(2,3)

zmienna systemowa DEFADD (2 bajty: 23563, 23564) wskazuje na obszar parametrów umieszczonych w pamięci:

x 14 ⟨5 bajtów⟩ y 14 ⟨5 bajtów⟩

wartość drugiego parametru
nazwa drugiego parametru
wartość pierwszego parametru
bajt o wartości 14 ozn. koniec nazwy
nazwa pierwszego parametru

Kolejne parametry o jednoliterowych nazwach umieszczone są począwszy od 3, 10, 17, itd. bajtu pola wskazywanego przez zmienną DEFADD.

Jeżeli wartość parametru jest liczbą całkowitą z przedziału od 0 do 65535, to jest ona zapamiętywana na pięciu bajtach w uproszczonej formie. W innym przypadku jest on zapisywany w formie liczb zamiennoprzecinkowych ZX Spectrum (patrz p. 2.5). Forma uproszczona ma następującą postać:

0,0, bajt mniej znaczący, bajt bardziej znaczący, 0.

W takim przypadku program asemblerowy łatwiej może uzyskać wartość parametru. Nie trzeba „przekodowywać" 5-bajtowej liczby zapisanej w specjalnym formacie.

Niżej podano procedurę, napisaną w języku asemblera, która oblicza różnicę symetryczną dwu bajtów (XOR)*) i wynik przesyła do pary rejestrów BC. Procedura ta została wykorzystana do zabezpieczenia programu napisanego w języku Basic przed skopiowaniem (p. 4.7). Założono, że parametry (dwa) są liczbami z zakresu 0...255. Są więc przekazywane w uproszczonej formie, przy czym bajt bardziej znaczący ma wartość zero. Program napisany w języku asemblera wygląda następująco:

*) Operator logiczny XOR : a XOR b daje wartość 1, gdy $a = 0$ i $b = 1$ lub $a=1$ i $b=0$, oraz wartość 0, gdy $a=0$ i $b=0$ lub $a=1$ i $b=1$ (patrz także 4.7.2).

Program	Kod binarny	Opis
LD IX,(23563)	221, 42, 11, 92	do rejestru IX ładowana jest zawartość zmiennej DEFADD
LD A,(IX+4)	221, 126, 4	do rejestru A ładowany jest pierwszy parametr
XOR A,(IX+11)	221, 174, 11	wykonywana jest operacja XOR z drugim parametrem
LD C,A	79	przekazanie wyniku w parze rejestrów BC
LD B,0	6, 0	
RET	201	

Program ten przetłumaczony na kod binarny i umieszczony dziesiętnie w instrukcji DATA ma postać:

```
10 DATA 221,42,11,92,221,126,4,221,174,11,79,6,0,201
```

Poniżej podano program, który wykorzystuje opisany podprogram umieszczony w buforze drukarki i definiuje funkcję obliczającą różnicę symetryczną dwu bajtów:

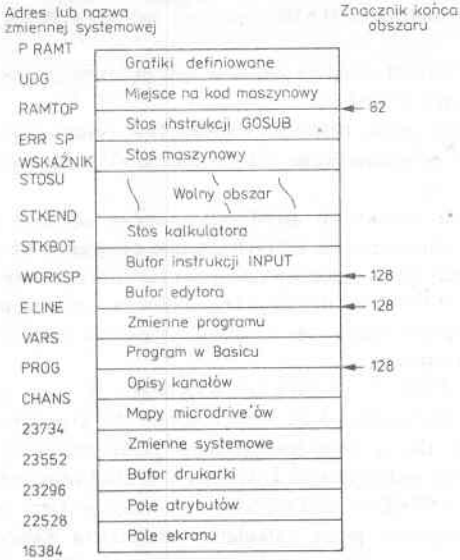
```
  10 DATA 221,42,11,92,221,126,4
,221,174,12,79,6,0,201
  20 FOR a=23296 TO 23296+13
  30 READ i: POKE a,i
  40 NEXT a: REM wczytanie do pa
mieci programu asemblerowego
  50 DEF FN x(y,z)=USR 23296
  60 INPUT "podaj dwie liczby ca
lkowite z  zakresu 0..255 ?",a,
b
  70 IF (a<0) OR (a>255) OR (a<>
INT a) OR (b<0) OR (b>255) OR (b
<>INT b) THEN GO TO 60
  80 PRINT a;"xor";b;"=";FN x(a,
b)
  90 GO TO 60
```

* * Oprogramowanie systemowe 4.5

Organizacja pamięci ZX Spectrum 4.5.1

Ze względu na przechowywanie informacji związanych z różnymi zadaniami realizowanymi aktualnie przez mikrokomputer jego pamięć RAM podzielono na kilka obszarów. Niektóre z nich, zawierające dane stale obecne w pamięci, mają ustalone niezmienne granice. Granice innych pól zmieniają się zgodnie z aktualnymi potrzebami.

Adresy początków i końców takich obszarów przechowują odpowiednie zmienne systemowe, wyróżnione bajty pamięci RAM znajdujące się w jej stałym polu. Niektóre zmienne obszary są rozgraniczone między sobą dodatkowym



Rys. 24. Mapa pamięci RAM ZX Spectrum

znakiem kontrolnym (na rys. 24 po prawej stronie po strzałce podano kod tego znaku).

Poszczególne obszary pamięci to (w nawiasach podano adres początku i końca obszaru lub nazwy zmiennych systemowych określających te adresy):

— pole ekranu (16384÷22527) — obszar używany do przechowywania informacji o stanie poszczególnych punktów obrazu (dokładniejsze informacje podano w rozdz. 3),
— pole atrybutów (22528÷23295) — przechowuje informacje o atrybutach kolejnych znaków ekranu (patrz rozdz. 3),
— bufor drukarki (23296÷23551) — używany jest do zapisywania linii 32 znaków przekazywanych do dołączonej drukarki. Znaki, podobnie jak na ekranie, mają postać ciągów kropek dokładnie odpowiadających ich obrazowi. Nie są zatem przechowywane w postaci kodów ASCII. Współpraca ZX Spectrum z drukarką polega na transmitowaniu do niej kolejnych, uformowanych rzędów kropek całej 32-znakowej linii. Gdy nie korzysta się z instrukcji LPRINT, COPY i LLIST obszar ten można wykorzystywać w innym celu, np. do przechowywania programów napisanych w języku wewnętrznym,
— zmienne systemowe (23552÷23733) — poszczególne bajty tego pola zawierają różne informacje ustalające sposób działania mikrokomputera (dokładny opis poszczególnych zmiennych systemowych zamieszczono w dodatku C, w następnym punkcie tego rozdziału przedstawiono ich podział i zastosowania),
— mapy microdrive'ów (23734÷CHANS) — zawarte są tu informacje o poszczególnych sektorach aktualnie używanej kasetki z taśmą. Jeżeli microdrive nie

jest podłączony, zmienna CHANS wskazuje na adres 23734 (pole to nie istnieje),

- opisy kanałów (CHANS ÷ PROG – 2) — w tym obszarze znajdują się rekordy kanałów (opis patrz p. 4.6),
- program w języku Basic (PROG ÷ VARS – 1) — przechowuje linie tekstu programu. Są one przechowywane w innej formie niż widoczna na ekranie (szczegóły w p. 2.5),
- zmienne programu (VARS ÷ E LINE – 2) — używa się tego pola pamięci do zapamiętania zmiennych, na których operuje program napisany w Basicu (informacje na temat formy przechowywanych tu informacji zawarto w p. 2.5.2).
- bufor edytora (E LINE ÷ WORKSP – 1) — bufor ten przechowuje aktualnie poprawianą lub wprowadzaną linię programu względnie zlecenie wykonywane w trybie natychmiastowym,
- bufor instrukcji INPUT (WORKSP ÷ STKBOT – 1) — do pewnej części tego obszaru są wpisywane dane wprowadzane do programu za pomocą instrukcji INPUT. Druga część jest używana przez interpreter Basicu jako obszar roboczy przy wykonywaniu kolejnych instrukcji programu,
- stos kalkulatora (STKBOT ÷ STKEND – 1) — między tymi adresami jest ulokowany stos używany przez kalkulator interpretera Basica. Kalkulator służy do obliczania wartości wyrażeń arytmetycznych. Przez stos są przekazywane argumenty obliczeń zmiennoprzecinkowych, wyrażeń tekstowych, itd. Działanie kalkulatora jest uruchamiane instrukcją kodu maszynowego RST 40 i po zakończeniu działania na szczycie stosu znajduje się wynik operacji. Kody poszczególnych instrukcji kalkulatora są różne od kodów operacji procesora Z80,
- wolny obszar (STKEND ÷ wskaźnik stosu procesora (SP)) — pole to jest wolne, lecz nieostrożna ingerencja może spowodować błędy działania systemu, w tym polu bowiem rozrasta się stos maszynowy,
- stos maszynowy (wskaźnik stosu procesora ÷ ERR_SP) — jest to stos używany przez procesor Z80 do przechowywania aktualnie przetwarzanych danych, adresów, itp.,
- stos instrukcji GO SUB(ERR SP + 1 ÷ RAMTOP) — przechowuje numery linii (i numery instrukcji w linii), do których należy wrócić po wykonaniu instrukcji RETURN. Informacja taka jest zawarta także na stosie maszynowym,
- miejsce na kod maszynowy (RAMTOP ÷ UDG) — pole to można wykorzystywać do zapamiętania programów napisanych w języku wewnętrznym, danych, itp. (metody zmian wielkości tego obszaru i ochrony przed skasowaniem podano w p. 2.4),
- grafiki definiowane przez użytkownika (UDG ÷ P RAMT) — ten obszar przechowuje obrazy znaków graficznych odpowiadających klawiszom od A do U w trybie G. Jego opis podano w punkcie 3.5.

O dynamicznym przydziale pamięci należy pamiętać szczególnie podczas prac prowadzonych w asemblerze. Granice poszczególnych obszarów od adresu

23734 do RAMTOP są bardzo często zmieniane. Wszystkie odwołania do danych zawartych w tych obszarach winny być adresowane według adresów przechowywanych w odpowiednich zmiennych systemowych.

Zmienne systemowe **4.5.2**

Kolejne bajty pola zmiennych systemowych określają sposób działania poszczególnych części składowych oprogramowania ZX Spectrum. Zmienne systemowe można w przybliżeniu podzielić na 4 klasy. Są to zmienne:
— określające granice poszczególnych części pamięci RAM,
— testujące aktualny stan klawiatury,
— stanu systemu,
— obsługi wejścia—wyjścia.

Ich dokładny wykaz zamieszczono w dodatku C. Poszczególne zmienne mają różną długość, w związku z tym w dodatku w pierwszej kolumnie podano liczbę bajtów przez nie zajmowaną. W ZX Spectrum przyjęto konwencję, że mniej znaczący bajt zmiennej dwubajtowej jest umieszczany w komórce o niższym adresie (istnieją wyjątki). Podana nazwa nie jest rozpoznawana przez interpreter Basica — wartości, które chce się wprowadzić do tego obszaru muszą być przekazywane instrukcją POKE.

Znaczenie wielu spośród tych zmiennych przedstawiono w innych rozdziałach książki. Wiele z pozostałych nie ma praktycznego zastosowania dla użytkownika mikrokomputera, względnie ich użycie wymaga bardzo dokładnej znajomości sposobu jego pracy. Poniżej omówiono wybrane zmienne razem z przykładami ich zastosowań, względnie wskazano źródło dokładniejszych informacji.

Zmienne określające granice poszczególnych części pamięci RAM

Zmienne te zostały przedstawione w poprzednim punkcie tego rozdziału. Wymieniona już zmienna

```
ERR SP 23613 (2)
       ↑     ↑
     adres  liczba bajtów
```

nie tylko rozgranicza stos maszynowy od stosu GO SUB. Pod tym adresem w trakcie wykonywania programu jest przechowywany adres procedury obsługi błędu i obsługi klawisza BREAK. Zmieniając zawartość stosu wskazywanego przez ERR SP można spowodować przejście programu do zupełnie innej obsługi błędu. Jest to sposób na rozszerzenie języka Basic ZX Spectrum o nowe instrukcje. Niestety jest to sposób trudny i wymagający uwzględnienia zmian zawartości stosu podczas pracy programu. Zmienna ta współpracuje z omówioną poniżej zmienną ERR NR.

Zmienna P RAMT 23732 (2) określa adres ostatniego bajtu dostępnej pamięci RAM. Dostępnej, a nie zainstalowanej, ze względu na sposób jej ustawiania przez systemowy program inicjalizacji. Program ten (tzw. Restart 0) rozpoczynający się od adresu 0 jest wykonywany po włączeniu zasilania. System przegląda całą pamięć RAM, poszukuje jej uszkodzenia podczas cyklu zapis — odczyt. Jeżeli wpisywana informacja zostanie przekłamana, to aktualnie wskazywany adres jest zmniejszany o 1 i ta wartość jest wpisywana do P RAMT.

Zmienne testujące aktualny stan klawiatury zostały opisane w rozdz. 1.

Zmienne stanu systemu

Zmienne stanu systemu określają adres i numer aktualnie wykonywanej instrukcji programu napisanego w Basicu, bądź instrukcji, do której należy przekazać sterowanie po zakończeniu wykonywania operacji, względnie aktualny stan interpretera Basica.

Zmienna MODE 23617 (1) określa rodzaj aktualnie drukowanego kursora, np. przy instrukcji INPUT. Wpisywanie w tę komórkę różnych wartości powoduje zmianę znaku kursora na inny np. 14 na „a" 236 na „?". Kursorowi można przywrócić normalną postać przyciskając klawisze CAPS SHIFT i SYMBOL SHIFT bądź GRAPHICS. Wpisywanie do tej komórki różnych liczb nie utrudnia jednak normalnego korzystania z klawiatury jako źródła danych.

Zmienne NEWPPC 23618(2) i NSPPC 23620(1) zawierają odpowiednio numer linii i numer instrukcji w linii, do której program w Basicu ma wykonać skok (przekazać sterowanie). Zmieniając je można uzyskać, tak jak w przykładowym programie, skok do linii o dowolnym numerze i dowolnej instrukcji w tej linii.

Linia 30 programu jest zatem (łącznie z linią 9999) ekwiwalentem instrukcji GOTO *numer linii, numer instrukcji w linii*:

```
  10 PRINT "Kasia"
  20 PRINT "-ma psa": PRINT "-ma
 jamnika Gucia"
  30 LET n=20: LET s=2: GO TO 99
99
9999 POKE 23618,(n-256*INT (n/25
6)): POKE 23619,INT (n/256): POK
E 23629,2
```

Zmienne OLDPPC 23662(2) i OSPPCC 23664(1) zawierają numer linii i numer instrukcji w linii, do której należy wrócić po przerwaniu wykonania programu. Są to parametry do zlecenia CONTINUE. W związku z tym wpisując w to pole inne wartości można wymusić skok do innego jego miejsca, np.: po zatrzymaniu pytaniem scroll? i wciśnięciu BREAK, N, SPACE, poniższego programu:

```
 5 LET i=1
10 LET i=i+1
20 PRINT i
30 GO TO 10
```

podanie POKE 23662,5: POKE 23664,1 : CONTINUE spowoduje ponowny wydruk liczb począwszy od jedynki.

Zmienna NXTLIN 23637 (2) podaje adres początku następnej instrukcji programu. Zmieniając ten adres można spowodować zaburzenie kolejności wykonywania programu. Jest to także metoda na eleganckie zakończanie programów asemblerowych lub przejścia z nich do Basica. Wpisując w formacie w jakim przechowywane są w pamięci instrukcje Basica dowolny rozkaz, np. 0 STOP (0,0,2,0,226,1,3) i ustawiając na adres tej instrukcji zmienną NXTLIN spowoduje się zakończenie podprogramu wywołanego przez USR komunikatem: STOP in statement 0 : 1.

Zmienna ERR NR 23610(1) zawiera pomniejszony o jeden numer błędu, który nastąpił podczas realizacji programu, a więc normalnie wpisane jest tu 255 (zerowy numer błędu oznacza jego brak). Zmienną tę można wykorzystywać we własnych procedurach obsługi błędu (łącznie ze zmienną ERR SP). Wpisanie w tę komórkę liczby z zakresu 0÷20 spowoduje, przy prawidłowym zakończeniu programu, wypisanie na ekranie komunikatu o błędzie zgodnym z opisanym numerem.

Zmienna DATADD 23639(2) wskazuje adres kolejnego pola DATA, które ma być wczytane instrukcją READ. Wadą READ jest możliwość ładowania wartości zdefiniowanych w DATA jedynie sekwencyjnie. W przypadku znajomości adresów kolejnych pól tej instrukcji można to zmienić. Oto prosty przykład:

```
  20 DATA "a","b","c","d","e"
  30 FOR i=1 TO 5
  40 READ k$: PRINT k$
  50 POKE 23639,202: POKE 23640,
92
  60 NEXT i
```

Linia 50 działa dokładnie tak samo jak instrukcja RESTORE. Wpisując inny adres do zmiennej DATADD (uprzednio trzeba sprawdzić jaki) można wczytywać różne pola do kolejnych zmiennych. Jest to szczególnie przydatne w programach edukacyjnych, gdzie stały zestaw pytań należy zadawać w losowej kolejności.

Zmienne obsługi wejścia—wyjścia

Kolejne bity zmiennych FLAGS 23611, TV FLAG 23612, FLAGS2 23658, FLAGX 23665, P FLAG 23697 są wykorzystywane przez system jako wskaźniki dotyczące źródła i miejsca przeznaczenia wysyłanych informacji, trybu działania programu, parametrów obrazu telewizyjnego. Zapalenie lub zgaszenie bitu określa jednoznacznie jaki jest stan związanej z nim operacji. Większość z tych wskaźników nie znajdzie praktycznego zastosowania przy programowaniu w Basicu.

Można stwierdzić, że FLAGS określa aktualny tryb pracy programu, TV FLAG — operacje na dolnej lub górnej części ekranu, FLAGS2 współpracuje z kanałami, z FLAGX korzysta system przy wprowadzaniu danych do systemu, a P FLAG określa INK, PAPER, OVER i INVERSE.

Co warto wiedzieć na ich temat? W programach, niekoniecznie pisanych w Basicu, w których naciśnięcie dowolnego klawisza ma spowodować jakąś reakcję, warto testować bit 5 zmiennej FLAGS. Po wstępnym wpisaniu do niego zera, jego zapalenie w trakcie wykonywania programu oznacza, że jakiś klawisz został naciśnięty. Z kolei gdy bit 7, najbardziej znaczący bit FLAGS, jest jedynką oznacza, że program jest w trakcie wykonywania, a jeżeli zawiera zero — analizowana jest przez system jego składnia, a więc program się nie wykonuje. A zatem taki program:

```
10 POKE 23611,0
20 PRINT "Ten program nic nie robi"
```

skończy swoje działanie już w linii 10.

Instrukcja POKE 23658,8 (zmiana FLAGS2) spowoduje zmianę kursora z L na C. Jest ona zatem równoważna naciśnięciu klawisza CAPS LOCK. Wykonanie POKE 23658,0 przywróci możliwość pisania małymi literami.

Znaczenie poszczególnych bitów pola P FLAG podano w p. 3.3. (rys. 13).

Liczba zawarta w zmiennej PIP 23609 (1) określa długość dźwięku emitowanego przy naciśnięciu klawisza. 0 oznacza ciche puknięcie, 255 długi dźwięk. Wprowadzenie do tej zmiennej wartości większej niż początkowa (0) pozwoli użytkownikowi upewnić się bez patrzenia na ekran, czy klawisz został wciśnięty.

Pierwszy z bajtów zmiennej COORDS 23677 (2) określa współrzędną x, a drugi współrzędną y, ostatnio napisanego punktu na ekranie. Zmiana zawartości tych komórek jest prostą i szybką metodą przesuwania początku wykresu na ekranie.

W prostym programie:

```
  10 FOR i=1 TO 11
  20 POKE 23677,100+i*4: POKE 23
678,100+i*4
  30 DRAW 20,5,PI*i/6
  40 NEXT i
```

Linia 20 jest równoważna instrukcji PLOT 100+i*y, 100+i*x.

Zmienna DF CC 23684 (2) określa adres dla następnej danej wyprowadzanej na ekran instrukcją PRINT. Sposób wyznaczania adresu dowolnego punktu na ekranie podano w rozdziale 3.

Poniższy prosty przykład pokazuje, że można nakładać na siebie różne napisy przesuwając pozycję drukowania:

```
10 PRINT AT 10,11 "Ala ma kota",:
   POKE 23684,72 : POKE 23685,68 : PRINT "kot ma Ale":
```

Zmienna S POSN 23688 (2) zapamiętuje aktualną pozycję wydruku na ekranie. Jej zawartość należy interpretować zgodnie z regułami:

liczba = 33 — PEEK (23688) określa numer kolumny,
liczba = 24 — PEEK (23689) — numer wiersza

aktualnej pozycji drukowania tekstu w instrukcji PRINT.

Zastosowanie procedur systemowych 4.5.3

System operacyjny jest gospodarzem komputera. Zarządza on jego działaniem, określa rodzaj wykonywanego działania i nadzoruje wykonanie programu, przydziela także kolejnym zadaniom rozmaite zasoby komputera — to jest procesor czy pamięć. W dużych komputerach współpraca z systemem operacyjnym odbywa się przez zlecenia podawane w specjalnym języku. W ZX Spectrum system jest nierozerwalnie związany z interpreterem Basica. Jego zlecenia stanowią instrukcje Basica. Cały system operacyjny zawarto w pamięci stałej.

Zawartość 16-kilobajtowej pamięci stałej ZX Spectrum można podzielić na cztery części:

- Procedury obsługi urządzeń zewnętrznych
- Interpreter Basica
- Kalkulator służący do obliczania wartości wyrażeń
- Generator znaków

Dokładniej analizując treść systemu można stwierdzić, że początkowe komórki zawierają tzw. restarty — procedury mające wiele zastosowań w poszczególnych częściach systemu. Dalej wpisano podprogramy obsługi przerwań i tablice słów kluczowych. Po tej wstępnej części rozpoczyna się blok procedur związanych z obsługą klawiatury, głośnika, magnetofonu, ekranu i drukarki. Następnie ROM zawiera programy systemowe sterujące interpretacją i wykonaniem programu napisanego w Basicu. Procedury wyznaczania wartości wyrażeń arytmetycznych i operacje na ciągach znaków zajmują kolejne części pamięci ROM. Kalkulator napisany w języku maszynowym, korzystający z własnego stosu i używający własnych instrukcji, przeprowadza wszystkie operacje arytmetyczne Basica, szczególnie zmiennoprzecinkowe. Ostatnie bajty pamięci stałej zajmuje generator znaków.

System ZX Spectrum jest, jak na możliwości oferowane przez ten mikrokomputer, bardzo zwarty, zajmuje tylko 16 kB. Z tego względu pewne procedury są używane do wielu zastosowań. Muszą być one uniwersalne, testować wiele parametrów, co odbija się na szybkości działania Basica Spectrum.

Opis działania i wykorzystania procedur systemowych przekroczyłby niewątpliwie założoną objętość niniejszej książki. Dlatego, tak jak przy opisie zmiennych systemowych, zostanie podanych kilka przykładowych procedur i kilka recept: jak i po co wykorzystywać procedury systemowe. Na ogół wymagają one podania pewnych parametrów, które muszą być załadowane bezpośrednio do rejestrów procesora, a więc mogą być one wykorzystywane jedynie z poziomu asemblera. W takich przypadkach przy opisie programów asemblerowych podano wartości bajtów, które należy wpisać do pamięci, np. instrukcją DATA, a następnie wykonać RANDOMIZE USR (*adres*).

Pewne zastosowania procedur współpracy z magnetofonem zostaną omówione w p. 4.7.

Restarty mogą być wywoływane jedną instrukcją RST *numer*. RST 0 — powoduje start procedury inicjalizacji, tak jak po włączeniu komputera, a więc wyzerowanie pamięci, ustawienie zmiennych systemowych na wartości początkowe (także RAMTOP), itd. RANDOMIZE USR 0 jest po prostu równoważne rozpoczęciu pracy od początku. RST 8 to często wykorzystywana procedura. Służy ona do przerywania wykonywania programu, gdy nastąpi błąd. Wywołanie:

Instrukcja	Kod binarny
RST 8	207
Numer błędu	Numer błędu

spowoduje wydrukowanie na ekranie komunikatu odpowiadającego podanemu numerowi błędu.

RST 16 to standardowy podprogram obsługujący instrukcję LIST i PRINT. Zgodnie z zapamiętanymi znacznikami (FLAGS, TV FLAGS) i numerem współpracującego kanału (CURCHL — patrz p. 4.6) wyprowadza znak z akumulatora procesora Z80 do urządzenia zewnętrznego. Na przykład:

LD A,78	załaduj do akumulatora kod znaku	62,78
RST 16	wykonaj restart 16	215
RET	powrót	201

spowoduje wysłanie znaku o kodzie 78 (N) na aktualnie używane urządzenie zewnętrzne (na ogół górna część ekranu).

Dwa następne restarty: RST 24 i RST 32, są używane przez interpreter Basica do pobierania do analizy kolejnych znaków z linii programu. RST 24 jest równoważny pobraniu aktualnie wskazywanego znaku (zmienna CH ADD), a RST 32 pobraniu następnego znaku i przesunięciu wskaźnika o jeden.

Restart RST 40 powoduje wywołanie programu kalkulatora. Komórka STKEND wskazuje argumenty operacji, a bajt następujący po rozkazie RST 40 definiuje jakie działanie należy wykonać, np.:

RST 40	239
1F	31

spowoduje obliczenie wartości sinusa argumentu znajdującego się na szczycie stosu i wpisanie wyniku na stos.

O dalszych zastosowaniach procedur systemowych trudno jest pisać w sposób systematyczny ze względu na ich liczbę i obszerność. W związku z tym dalsze uwagi na ten temat zostaną przedstawione w postaci recept jak je używać przy pisaniu programów w Basicu (wg „Personal Computer World" i innych źródeł).

Przesuwanie linii na ekranie, czyli scrollowanie, można zrealizować za pomocą wywołania RANDOMIZE USR 3582, które jest bezpośrednim odwołaniem do systemowej procedury CL_SC_ALL wykorzystywanej przy scrollowaniu. Przesuwać można nie tylko z dołu do góry całą zawartość ekranu, tak jak w programie:

```
 5 FOR i=1 TO 32*6-4
10 PRINT "Kasia"
20 RANDOMIZE USR 3582
30 NEXT i
```

Można, na przykład, przesuwać tylko dolną połówkę ekranu. W tym celu należy wykonać RANDOMIZE USR 3583 (wywołanie tej samej procedury). Ginie środkowa linia wypisanego tekstu, a od dołu pojawia się nowa. Dolną połowę tekstu można skasować. USR 3652 odwołuje się do podprogramu, w którym liczba linii dolnej części ekranu przeznaczonych do skasowania jest przesyłana w rejestrze B procesora. Zatem można skasować dowolną liczbę linii za pomocą podprogramu:

```
LD B, N      0,6, liczba linii
CALL 0E44    205, 68, 14
RET          201
```

Wywołanie podprogramu z adresu 3330 da Czytelnikowi możliwość przesunięcia całej strony tekstu na ekranie naraz. Wywołanie powinno mieć postać RANDOMIZE USR 3330. Program można także zatrzymać w dowolnej chwili działania za pomocą wywołania funkcji: USR 3213, która odwołuje się do procedury sterującej przesuwaniem linii w taki sposób jakby cały ekran był już pełny i drukuje w ostatniej linii ekranu pytanie scroll?

Doświadczony użytkownik ZX Spectrum, który choć trochę próbował pisać w asemblerze i wywoływać wewnętrzne procedury dobrze zna taki widok: po wywołaniu programu ekran czernieje i po chwili pojawia się napis: © 1982 Sinclair Research Ltd. Znaczy to, że podobnie jak po włączeniu zasilania, pracę trzeba zacząć od początku. Sam ten napis można jednak uzyskać bez zerowania pamięci — RANDOMIZE USR 4757 pozwoli go stosować w każdej chwili. Uogólniając, na ekranie można uzyskać dowolny spośród komunikatów błędów lub słów kluczowych Basica, podając w rejestrach DF adres początku tablicy komunikatów lub słów kluczowych, a w A — numer błędu lub kod znaku i wywołując procedurę o adresie 3082.

Na zakończenie parę słów o efektach specjalnych. Niektóre z metod uzyskiwania niekonwencjonalnego obrazu na ekranie już podano. Z rozdziału o grafice wiadomo, jak zmieniać kolor brzegu ekranu (BORDER). Ale żeby był on w paski, tak jak przy wprowadzaniu danych z magnetofonu? Warto wypróbować:

```
10 PAPER 1:INK 7: RANDOMIZE USR 1333: CLS
20 REM reszta programu
```

Dźwięk i przesuwanie pasków można zatrzymać naciśnięciem spacji. Podany adres (1333) to jeden z punktów wejścia do wewnętrznej pętli podprogramu SAVE. Kolory pasów brzegu ekranu można zmieniać odpowiednio wywołaniem:
1331 — magenta i niebieskie (tzn. różowofioletowe i niebieskie),
1290 — niebieskie i czerwone,

1269 — kolory jak poprzednio, ale za to z dźwiękiem,
1251 — zielone i magenta,
1248 — czarne i białe,
1333 — cyan i niebieskie,
1327 — niebieskie i białe,
1367 — niebiesko-żółte.

Testowanie użycia innych procedur pozostawiamy Czytelnikom. Do tej zabawy trzeba mieć trochę czasu i cierpliwości, ale uzyskany na ekranie efekt (nawet o ile to będzie ponowny start systemu) może wynagrodzi te żmudne starania.

* * * Koncepcja strumieni i kanałów. Sposób definiowania kanałów 4.6

S t r u m i e ń d a n y c h można sobie wyobrazić jako urządzenie, które potrafi przesłać do komputera ciąg znaków lub wczytać ciąg znaków z komputera. Strumień danych jest wyimaginowanym, lecz bardzo prostym w obsłudze urządzeniem.

W ZX Spectrum realizowano mechanizm, który pozwala „podłączyć" strumień danych do konkretnego urządzenia zewnętrznego. Może to być dowolne urządzenie, np.: drukarka, klawiatura, ploter, itd. Takie podłączenie wraz z urządzeniem zewnętrznym nazywa się k a n a ł e m. Zaletą strumieni jest uniezależnienie programu od urządzenia, z którym współpracuje. Sprzężenie programu z konkretnym urządzeniem następuje poza programem, a realizuje je system operacyjny.

W ZX Spectrum można używać szesnastu strumieni identyfikowanych numerami od 0 do 15. Wszystkich strumieni używa się w ten sam sposób.

Podstawowymi instrukcjami współpracy ze strumieniami danych są:

INPUT #*n*; *lista* — do wprowadzenia danych ze strumieni do komputera,

PRINT #*n*; *lista* — do wprowadzenia danych z komputera do strumienia.

↑ ↑ numer kanału
znak # oznacza, że instrukcja dotyczy kanału;
informacje (lista) składają się ze zmiennych, stałych liczb oraz tekstów

Na przykład:

```
10 PRINT #3; A;B
```

powoduje przesłanie wartości zmiennych A i B do strumienia danych nr 3, natomiast:

```
15 INPUT #1; A;B
```

powoduje wczytanie ze strumienia pierwszego dwu wartości i podstawienie ich za zmienne A i B.

Mechanizm strumieni pozwala wprowadzać lub wyprowadzać dane za pomocą instrukcji INPUT i PRINT nie tylko z klawiatury i na ekran, lecz do dowolnego urządzenia zewnętrznego, np.: drukarki, dysku itd.

Każdy strumień składa się faktycznie z dwu strumieni. Strumienia wejściowego i wyjściowego. Dlatego możliwe jest użycie np. instrukcji typu:

10 INPUT #2 ; "Jak ci na imię?" ; A$

Instrukcja ta najpierw wysyła informacje do strumienia danych nr 2, a następnie wczytuje z niego dane.

Instrukcja INPUT i PRINT korzysta ze strumienia wyjściowego, a INPUT korzysta także ze strumienia wejściowego. Możliwe jest także użycie jednej instrukcji INPUT lub PRINT do współpracy z kilkoma strumieniami, np.:

10 PRINT #3; "strumień 3"; #4; "strumień 4"

Tekst „strumień 3" przesłany zostanie do strumienia nr 3, a tekst „strumień 4" do strumienia nr 4.

Związanie strumienia danych z konkretnym kanałem jest realizowane za pomocą instrukcji języka Basic:

OPEN #S, "K"*)

↑ ↑

nazwa kanału (pojedyncza litera)

numer strumienia (liczba z zakresu 0÷15)

Po włączeniu zasilania (lub wciśnięcia klawisza RESET) w fazie inicjacji systemu operacyjnego są otwierane strumienie o numerach od 0 do 3 i łączone tym samym z następującymi kanałami:

kanał	nazwa	
0	„K"	klawiatura
1	„K"	klawiatura
2	„S"	ekran
3	„P"	drukarka

Klawiatura „K" jest jedynym z wyżej wymienionych kanałów, który może transmitować dane w obu kierunkach — z i do komputera. Jako urządzenie wejściowe służy klawiatura, a urządzeniem wyjściowym jest dolna część ekranu (okno systemowe).

Kanały „P" i „S" mogą być używane wyłącznie jako urządzenia wyjściowe. Próba wczytania informacji z ekranu lub drukarki, np.:

INPUT #3;A

kończy się komunikatem o błędzie. Ponieważ kanał „K" (podłączony do stru-

*) Duże i małe litery w nazwie kanału mają takie same znaczenie, np. „K" jest równoważne „k".

mienia danych nr 0 i 1) umieszcza informacje w oknie systemowym na ekranie telewizora, to można drukować w tej części ekranu zarówno za pomocą instrukcji:

PRINT # 0; "To jest okno systemowe"
PRINT # 1; "To jest także okno systemowe"

jak i:

INPUT # 1; "To jest okno systemowe"

„Odłączenie" strumienia danych od kanału realizuje instrukcja:

```
CLOSE # S
        ↑
        numer kanału
```

Zamknięty (odłączony) kanał może zostać otwarty (przydzielony) dla innego strumienia danych, np.:

```
 10 OPEN #5,"s": PRINT #5;"gora
ekranu"
 15 PAUSE 50
 20 CLOSE #5
 30 OPEN #5,"k": PRINT #5;"klaw
iatura-dol ekranu": PAUSE 50
```

Strumień danych współpracujący początkowo z ekranem współpracuje następnie z drukarką.

Uwaga *Niemożliwe jest zamknięcie żadnego ze standardowych (predefiniowanych) kanałów (tzn. „K", „P", „S"). Zamknięcie takiego kanału powoduje automatyczne jego otwarcie dla strumienia danych o numerze standardowym. Instrukcja RUN zamyka wszystkie kanały oprócz predefiniowanych.*

W ZX Spectrum standardowo można używać tylko trzech kanałów. Jeśli podłączymy do komputera układ Interface 1 oraz microdrive, to można używać także kanałów „M", „N", „T" i „B". W przypadku kanału „M" każdy strumień związany jest z innym plikiem danych na kasetce microdrive'u. W instrukcji OPEN podaje się wtedy dodatkowo nazwę pliku (patrz rozdz. 6). W pamięci ROM komputera znajdują się gotowe procedury (napisane w języku asemblera) umożliwiające współpracę z klawiaturą, ekranem i drukarką. W pamięci stałej zawartej w układzie Interface 1 umieszczono m.in. procedury obsługi microdrive'u.

Jeśli chcemy podłączyć do ZX Spectrum nietypowe urządzenie, to musimy sami oprogramować kanał. W dalszej części tego rozdziału problem ten został wyjaśniony dokładnie.

INPUT i PRINT nie są jedynymi instrukcjami współpracy ze strumieniami danych. Treść programu w Basicu (tzw. program źródłowy) można przesłać do dowolnego strumienia za pomocą instrukcji:

```
LIST # S,n
       ↑ ↑
       │ numer linii, od której należy rozpocząć transmisję
       numer strumienia
```

Uwaga *LIST # S oznacza przesłanie do strumienia danych całego programu.*

Pojedynczy znak można wczytać ze strumienia danych za pomocą funkcji INKEY$, np.:

```
PRINT INKEY$ #3
               ↑
               numer strumienia
```

Funkcja INKEY$ nie oczekuje na dostarczenie znaku, podobnie jak ma to miejsce, gdy używamy jej do współpracy z klawiaturą. Jeśli znak jest niedostępny, to wynikiem działania funkcji INKEY$ jest tekst pusty (a więc na ekranie nic się nie wydrukuje, nastąpi zmiana wiersza).

W obszarze zmiennych systemowych znajduje się pole STRMS (komórki pamięci o adresie od 23568 do 23605) długości 38 bajtów. W polu tym znajdują się informacje z jakim kanałem jest sprzężony każdy strumień oraz czy jest on otwarty. Dla każdego strumienia zapisano tam adres grupy bajtów opisujących kanał związany z tym strumieniem (tzw. rekord kanału). Dokładnie — nie adres kanału, ale liczbę, którą należy dodać do adresu początku obszaru, w którym zapamiętane są rekordy kanałów. Jeszcze dokładniej — dodać należy liczbę mniej 1. Jeśli wpisana liczba równa się 0, to strumień nie został otwarty (nie może być on użyty).

W ZX Spectrum można używać 16 strumieni o numerach od 0 do 15. Adres zajmuje 2 bajty: $16 \times 2 = 32$. Skąd więc 38 bajtów w polu STRMS? W polu tym zawarte są także informacje potrzebne do pracy trzech dodatkowych strumieni niedostępnych dla programisty używającego języka Basic. Strumienie te używane są przez system operacyjny. Współpracują one z następującymi kanałami:

„R" — kanał obsługujący bufor edytora i instrukcji INPUT.
„S" — ekran,
„K" — klawiatura.

Adresy kanałów współpracujących z trzema systemowymi kanałami zajmują pierwsze sześć bajtów pola STRMS. Adres kanału dla strumienia nr 0 zajmuje bajt 7 i 8, itd. W fazie inicjacji systemu operacyjnego pole to jest wypełniane w ten sposób, że wpisywane są tam adresy rekordów kanałów (kolejno kanału: „S", „R", „K", „K", „K", „S", „P"), współpracujących ze strumieniami systemowymi o numerach $0 \div 3$, oraz zera dla pozostałych strumieni.

Pole STRMS opisuje strumienie. Definicje (rekordy) kanałów są umieszczone natomiast w obszarze pamięci począwszy od adresu wskazywanego przez zmienną systemową CHANS (dwa bajty o adresach: 23631, 23632). Zajmują one obszar aż do programu w Basicu — czyli do miejsca wskazywanego przez zmienną PROG (bajty o adresie (23635,23636).

W czasie inicjalizacji systemu operacyjnego w pole wskazywane przez CHANS są wpisywane rekordy kanałów strumieni standardowych: „K", „S", „R", „P". Zmienna CHANS zawiera wtedy wartość 23734. Każdy rekord kanału zajmuje 5 bajtów. Pole CHANS zajmuje więc 20 bajtów. Struktura rekordu jest następująca:

2 bajty — adres procedury transmisji pojedynczego znaku z komputera do kanału (używanego, np.: w instrukcji PRINT #, LIST #)

2 bajty — adres procedury transmisji pojedynczego znaku z kanału do komputera (używanego, np. w instrukcji INPUT #, INKEY$ #)

1 bajt — nazwa kanału — pojedyncza litera (w kodzie ASCII).

Jeśli wykona się program:

```
10 LET a=23631
20 FOR i=a TO a+20 STEP 5
30 PRINT CHR$ (PEEK (i+4));" "
;PEEK (i+1)*256+PEEK i,PEEK (i+3
)*256+PEEK (i+2)
40 NEXT i
```

to na ekranie pojawi się struktura rekordów standardowych kanałów:

```
K 2548 2342
S 2548 5622
R 3969 5507
P 2548 5622
  ↑    ↑
  |    adres procedury wejścia
adres procedury wyjścia
```

Mechanizm działania kanałów można sobie wyobrazić następująco. Na podstawie zawartości pola STRMS interpreter znajduje adres rekordu kanału. W czasie wykonywania instrukcji współpracy ze strumieniem (np. PRINT) adres ten jest zapamiętany w komórkach o adresie 23633, 23634, tzn. zmiennej systemowej CURCHL (ang. *CURrent CHaneL*). Następnie dane (teksty lub liczby) są zamieniane na ciąg znaków zapisanych w kodzie ASCII. Ciąg ten jest transmitowany znak po znaku. Do przesłania znaku używa się odpowiedniej procedury transmisji znaku w kanale. Adres tej procedury pobiera się z rekordu kanału.

Strumienie danych i kanały są idealnym sposobem podłączenia do ZX Spectrum nowych urządzeń zewnętrznych. Dla takiego urządzenia należy napisać (w języku asemblera) procedurę transmisji znaków z i do komputera oraz zdefiniować rekord kanału.

Procedury transmisji znaku powinny spełniać następujące warunki:

— procedura transmisji znaku z komputera do urządzenia jest wywoływana przez interpreter Basica w ten sposób, że w akumulatorze A jest przekazywany znak, który ma być wysłany (znak zapisany w kodzie ASCII),

— procedura wczytania znaku do komputera powinna zwracać w akumulatorze A wczytany znak zapisany w kodzie ASCII. Wskaźnik C powinien być

ustawiony na wartość 1. Jeżeli znak jest niedostępny (nie został wczytany z urządzenia), to wskaźniki C i Z (zero) powinny być wyzerowane,
— jeżeli oprogramowywane urządzenie nie jest zdolne prowadzić transmisji z i do komputera (zarówno czytać, jak i pisać np. drukarka), to zamiast adresu procedury przesłania znaku w niewłaściwym kierunku, np. czytania z drukarki, należy w rekordzie kanału umieścić adres procedury sygnalizującej błąd. Można skorzystać z procedury systemowej drukującej na ekranie komunikat o błędzie i zwracającej sterowanie do interpretera Basicu — RST 8. Wywołanie tej procedury zapisane w języku asemblera ma postać:

```
program            kod binarny

ERROR RST 8             8
      DEFB 18           18
           ↑
           kod błędu 18 oznacza: błędne urządzenie
```

Najprostszym sposobem dołączenia nowego urządzenia jest podmiana rekordu jednego z kanałów standardowych nowym rekordem. Należy napisać dwie procedury transmisji znaku (lub sygnalizacji błędu), a w polu CHANS w rekordzie jednego z kanałów standardowych wpisać, np. za pomocą POKE, adresy tych procedur. Wtedy nowy kanał będzie używany zamiast podmienionego. Nazwa kanału nie ulega zmianie.

Rozstrzygnięcia wymaga problem, który ze standardowych kanałów usunąć. Nie można usunąć kanałów ,,R" i ,,K" (ze względu na działanie systemu operacyjnego). Pozostaje więc wybór, czy usunąć kanał ,,S" (ekran), czy ,,P" (drukarkę). Najczęściej można zrezygnować z drukarki. W ten sposób można także podłączyć do ZX Spectrum niestandardową drukarkę.

Jeśli nie można zrezygnować z żadnego z kanałów predefiniowanych, należy napisać nowy rekord kanału. Można umieścić go w obszarze CHANS, ale konieczne jest wtedy zrobienie wolnego miejsca na rekord kanału. Można to uczynić za pomocą systemowej procedury MAKESP. Procedura MAKESP przesuwa używany przez interpreter Basicu obszar pamięci RAM w ten sposób, by zwolnić żądaną liczbę bajtów pamięci. Zmienia odpowiednio wartości wszystkich zmiennych systemowych, które powinny być zmienione w związku z reorganizacją pamięci.

Niżej podano program w asemblerze rezerwujący w polu CHANS miejsce dla nowego rekordu kanału.

program	kod	opis
LD HL,(23635)	42,83,92	do HL wpisz adres obszaru programu
DEC HL	43	23635 — adres zmiennej PROG
LD BC,5	1, 5, 0	Rezerwacja 5 bajtów
CALL 5717	205,85,22	Wywołanie MAKESP
RET	201	Powrót

Oto program w języku Basic, który rezerwuje pamięć w obszarze między ostatnim zajętym bajtem pola CHANS a bajtem, w którym rozpoczyna się program w Basicu. Program ten wykorzystuje napisany wyżej podprogram umieszczony w buforze drukarki.

```
  10 INPUT "ile bajtow zarezerwo
wac? ";l
  20 LET l1=INT (l/256): REM baj
t bardziej znaczacy
  30 LET l2=l-256*l1: REM bajt m
niej znaczacy
  40 LET a=23296: REM a umiesz
czono adres programu maszynowego
  50 FOR i=0 TO 10
  60 READ b: POKE i+a,b
  70 NEXT i: REM wpisanie do pam
ieci programu maszynowego
  80 LET a=23301
  90 POKE a,l2: POKE a+1,l1
 100 RANDOMIZE USR 23296
 110 DATA 42,83,92,43,1,5,0,205,
85,22,201
```

Linie 50, 60, 70 realizują wpisanie do pamięci programu asemblerowego umieszczonego w zbiorze DATA. W linii 90 następuje zmiana liczby bajtów, które będą przekazane parze BC jako parametr procedury MAKESP.

Podany program po wykonaniu należy usunąć. Niestety nie można tego zrobić za pomocą NEW, bo instrukcja ta odtwarza początkową wielkość pola CHANS. Trzeba usuwać go linia po linii.

Nowy rekord kanału może być również umieszczony w innym miejscu pamięci RAM. Jeśli rekord kanału nie jest umieszczony w polu wskazywanym przez zmienną systemową CHANS, to kanał ten nie może być otwarty za pomocą instrukcji OPEN. W czasie wykonania OPEN interpreter przeszukuje tylko obszar CHANS w poszukiwaniu kanału o podanej w tej instrukcji nazwie. Kanału takiego można jednak używać, jeśli wypełni się odpowiednio tablicę STRMS, np.: za pomocą instrukcji POKE. Jest jednak jeden warunek. Rekord kanału powinien być umieszczony w pamięci poniżej bufora danych używanego przez instrukcję INPUT, tzn. wskazywanego przez zmienną systemową WORKSP. W innym przypadku może dojść do zawieszenia systemu.

Praktycznie więc jedynym obszarem pamięci, w którym można umieścić rekord kanału jest bufor drukarki (jeżeli nie używa się instrukcji LLIST, LPRINT, COPY oraz żadnej z instrukcji korzystającej z kanału ,,P") lub pole CHANS.

Połączenie układu Interface 1 oraz microdrive zmienia sposób działania kanałów, gdyż inną postać ma wtedy rekord kanału. Rekord ten ma w takim przypadku zmienną długość, np. rekord kanału ,,M" liczy 595 bajtów i zawiera wszystkie informacje, które są potrzebne do zapisania bloku na taśmie microdrive'u, m.in. nagłówek i bufor danych.

Jeśli kanał zdefiniowany przez użytkownika ma działać zarówno wtedy, gdy Interface 1 jest odłączony, jak i podłączony, to rekord powinien mieć postać:

2 bajty — adres procedury wyjściowej,
2 bajty — adres procedury wejściowej,
1 bajt — nazwa,
2 bajty — o wartości: 64,0 — adres procedury obsługi błędu, znajdującej się w pamięci stałej układu Interface 1,
2 bajty — o wartości: 64, 0,
2 bajty — o wartości: 11, 0 — liczba bajtów w rekordzie kanału.

Do 5 bajtów stanowiących zawsze treść rekordu należy więc dodać sześć bajtów o ustalonych wartościach: 64, 0, 64,0 11,0. Dokładniejszy opis znajduje się w rozdz. 6.

* * Zabezpieczanie programów i łamanie zabezpieczeń 4.7

W dużych komputerach można dosyć dobrze zabezpieczyć programy przed odczytaniem lub używaniem przez niepowołane osoby. Programy zapisane na taśmie magnetofonowej nie mogą być skutecznie chronione, choć producenci oprogramowania stosują kilka trików, które utrudniają ich skopiowanie (mniej lub bardziej skutecznie).

Najprostszym sposobem skopiowania jest przegranie taśmy z programem z jednego magnetofonu na inny, tak jak kopiuje się piosenki. Nie jest to sposób niezawodny, lecz jeśli dysponuje się dobrymi magnetofonami i dobrą taśmą, zupełnie wystarczający. Nie istnieje sposób zabezpieczenia taśmy z programem przed skopiowaniem tą metodą.

Sposób zapisu programu na taśmie 4.7.1

Gniazda przeznaczone do współpracy z magnetofonem MIC oraz EAR są dołączone za pośrednictwem ULA do procesora jako układ wejścia—wyjścia o adresie 254 (tego samego, który steruje dźwiękiem oraz kolorem brzegu ekranu). Trzeci bit szyny danych jest podłączony do gniazda MIC, a szósty bit do EAR. Wykonanie programu:

```
10 OUT 254,0
20 OUT 254,8
30 GO TO 10
```

powoduje wysłanie na magnetofon na przemian zer i jedynek (8 odpowiada jedynce na trzecim bicie). Na wyjściu MIC pojawia się na przemian napięcie 1,3 V oraz 0,75 V.

Podobnie wykonanie instrukcji:

```
10 PRINT IN 254
20 GOTO 10
```

pozwala obejrzeć, czy przez wejście EAR odczytuje się z magnetofonu 0 czy 1 (napięcie niskie oznacza 0, wyższe 1). Jeśli szósty bit będzie zawierał 1, to na ekranie pojawi się liczba 255, gdy 0 — liczba 191.

Zapisywanie informacji na taśmie przypomina generację dźwięku (patrz rozdz. 5). Ponieważ szybkość przesuwania taśmy nie jest stała, a konieczna jest synchronizacja pracy komputera i magnetofonu nie można zapisywać wprost jedynki logicznej jako wysokiego napięcia, a zera logicznego jako napięcia niskiego. Użyto więc tzw. kodu samosynchronizującego. Każdy bit (zarówno 0 jak 1) jest zapisywany w ten sposób, że napięcie na wyjściu zmienia się z niskiego na wysokie (z 0,75 V na 1,3 V). Różny jest czas trwania stanów napięcia dla jedynki i dla zera. Dla jedynki jest on dwa razy dłuższy niż dla zera.

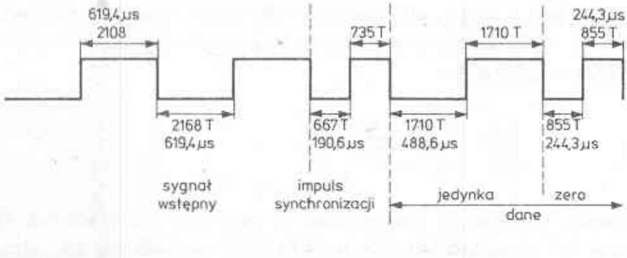
Wszystkie programy są zapisywane na taśmie w dwu blokach: krótki blok to nagłówek, dłuższy — blok programu lub danych. Bloki tych dwu typów mają na taśmie identyczny format. Na jego początku słychać jednostajny dźwięk — jest to wstępny, wiodący sygnał (trwa on 5 s przed nagłówkiem i ok. 2 s przed blokiem danych). W czasie trwania tego sygnału na wyjściu pojawia się na przemian niskie i wysokie napięcie, a każdy ton trwa 619 μs (to jest 2168 okresów zegara). Jest on używany po to, aby użytkownik mógł rozpoznać, że zaczyna się program, oraz by mogły ustawić się układy automatyki w magnetofonie. Dla komputera sygnał ten jest zbędny. Następnie pojawia się krótki (niesłyszalny) impuls synchronizacji. Informuje on komputer, że zaczynają się dane. Ton synchronizujący to niski stan przez 190 μs i wysoki przez 210 μs.

Bity danych są kodowane w następujący sposób:

— jedynka — niski, a następnie wysoki stan napięcia o czasie trwania 488,6 μs (to jest 1710 taktów zegara),
— zero — niski oraz wysoki stan napięcia przez okres 244,3 μs (to jest 855 taktów) każdy.

Sposób zapisu na taśmie przedstawiono na rys. 25.

Nagłówek zawiera wiele informacji koniecznych do prawidłowego odczytania bloku programu:



Rys. 25. Sposób zapisu sygnału na taśmie magnetofonowej

— typ programu:

0 — program w Basicu,
1 — tablica liczb,
2 — tablica tekstu,
3 — program binarny lub obraz binarnej pamięci ekranu,

— nazwa programu,
— długość (w bajtach),
— dla programów samostartujących w Basicu (typ = 0) numer linii, od której należy rozpocząć wykonanie, dla programów binarnych (typ = 3) adres ładowania,
— długość programu (pole istotne tylko dla programów typu 0).

Strukturę nagłówka przedstawiono na rys. 26. Ma on 17 bajtów długości.

Numer bajtu	0	1	11	13	15
	Typ	Nazwa	Długość	Adres lub długość linii	Długość programu w Basicu

Rys. 26. Struktura nagłówka

Na podstawie informacji zawartych w nagłówku komputer wczytuje blok danych.

W systemie operacyjnym ZX Spectrum są zawarte programy, które wykonują załadowanie bloku (także nagłówka) z taśmy lub zapisują go na magnetofon. Program ładujący rozpoczyna się od adresu 1366 (0556 heksadecymalnie), a program zapisujący na taśmie jest umieszczony począwszy od adresu 1218 (04C2 heksadecymalnie). Przed wywołaniem tych programów do rejestrów procesora należy załadować parametry:

DE — liczba bajtów, które należy wczytać lub zapisać (dla nagłówka 17),
IX — adres pierwszego bajtu w pamięci przeznaczonego do transmisji lub adres pola, w które zostaną wpisane dane (także nagłówek),
A — liczba 0 dla nagłówka, 255 dla bloku.

Wskaźnik C — przy odczycie oznacza: 1 — ładowanie do pamięci, 0 — weryfikowanie, przy zapisie znaczniki — C i Z są używane do sygnalizacji, czy ładowanie programu odbywało się bezbłędnie:

C Z
0 1 — błąd ładowania
0 0 — zły typ zbioru
1 1 — ładowanie poprawne

Opisane tu procedury systemowe są wykorzystywane przez programy kopiujące.

Program można wczytać do komputera stosując systemową procedurę ładowania. W tym celu należy wykorzystać poniższy program asemblerowy.

program	kod
LD DE, długość bloku	17, mniej znaczący bajt, bardziej znaczący bajt długości,
LD A, 0 lub 255	62,0 dla nagłówka 255 dla bloku
LD IX, adres	221,33, mniej, bardziej znaczący bajt adresu,
SCF	55
CALL 1366	205,86,5
RET	201

Sposób ładowania i uruchamiania takich podprogramów z Basica podano w pierwszych punktach tego rozdziału.

Zabezpieczanie programów i łamanie zabezpieczeń 4.7.2

Jedną z metod zabezpieczania programów przed kopiowaniem jest spowodowanie, by program zaraz po załadowaniu rozpoczynał pracę i nie dawał się w jej trakcie przerwać.

W ZX Spectrum można napisać w języku Basic program samostartujący (zapisany na taśmie instrukcją SAVE LINE) — patrz rozdz. 1 — lecz program taki daje się przerwać. Program napisany w Basicu daje się przerwać naciśnięciem klawisza BREAK. Klawisz ten można zablokować, w tym celu należy zawartość zmiennej systemowej ERR SP zmniejszyć o 2. Jeżeli jednak w programie napisanym w Basicu z zablokowanym w ten sposób BREAK wystąpi błąd, może dojść do zawieszenia systemu (zamiast wydruku komunikatu o błędzie). Jeżeli jednak program taki załadujemy do pamięci instrukcją MERGE (zamiast LOAD) nie wystartuje on samoczynnie i można go będzie zarówno wydrukować, jak i skopiować*).

Program w Basicu można zabezpieczyć przed listowaniem na ekran przez POKE 23755,100 (przywrócenie przez wpisanie w ten adres zera).

Program napisany w języku asemblera lub zapisany w języku wyższego rzędu (np. Pascalu czy Basicu) i przetłumaczony na kod wewnętrzny nie daje się przerwać. W Basicu ZX Spectrum nie ma standardowej możliwości zrobienia samostartującego programu binarnego. Wymyślono jednak inny sposób. Niech program napisany w asemblerze będzie umieszczony w pamięci począwszy od adresu 30000 i niech zajmuje 3000 bajtów. Należy wpisać do komputera, a następnie uruchomić program:

```
 5 CLEAR 33000
10 SAVE "nazwa" CODE 23552,9448
20 RANDOMIZE USR 30000
```

*) Sposób taki jest nieskuteczny, gdy korzysta się z microdrive'u.

W wyniku wykonania tych linii zostanie zapisany na taśmie program o podanej nazwie, który załadowany instrukcją LOAD "*nazwa*" CODE (lub LOAD"" CODE) do pamięci komputera wystartuje automatycznie i nie da się przerwać.

Uwaga *W instrukcji SAVE "nazwa" CODE adres zawsze musi wynosić 23552, a długość zapisywanego programu powinna być tak dobrana, by zapamiętany został zarówno program w asemblerze, jak i cały obszar pamięci dostępnej dla Basica (do komórki o adresie zawartym w zmiennej RAMTOP).*

Sposób działania podanego wyżej programu jest następujący: w linii 10 zapisuje się na taśmie obszar zmiennych systemowych (rozpoczynający się w miejscu pamięci o adresie 23552), obszar programu w Basicu wraz ze stosem interpretera Basicu (do adresu wskazywanego przez RAMTOP) oraz program maszynowy. Zmienne systemowe zapamiętywane są na taśmie w momencie, gdy interpreter powinien wykonać instrukcję z linii 20.

Po zakończeniu zapisywania rozpoczyna się wykonywanie programu maszynowego, lecz jest to już nieistotne. Można wyłączyć komputer. Program został już zapamiętany na taśmie. Jeżeli zapamiętany według tej metody program zostanie wpisany do pamięci komputera (LOAD""), to zmienne systemowe zostaną zmienione tak, że interpreter stwierdzi, że znajduje się w linii 20 podanego wyżej programu. Wykonuje się tę linię przekazując w ten sposób sterowanie do programu maszynowego.

Tak mozolnie zabezpieczony program można jednak skopiować. Jeżeli program jest dostatecznie krótki, można zmniejszyć pamięć dostępną dla Basicu (zmieniając zawartość RAMTOP) tak, aby cały zapisany na taśmie ciąg bajtów załadować w obszar niedostępny dla interpretera (ponad zmienną RAMTOP). Uniemożliwia to automatyczny start programu.

Gdy jednak program jest na tyle długi, że nie mieści się w pamięci, nawet gdy zmienna RAMTOP ma najmniejszą możliwą wartość, to należy spowodować błąd w czasie ładowania programu. Interpreter może zasygnalizować błąd, jeżeli poda się zbyt dużą długość bloku przeznaczonego do odczytania.

Często programy składają się z kilku fragmentów (tzw. nakładek). Pierwszy fragment załadowany do pamięci i uruchomiony wykonuje pewne czynności w czasie, gdy taśma w magnetofonie nadal się przesuwa, a na zakończenie ładuje następny fragment itd. Taką metodę można wykorzystać przy zapisywaniu programów wymagających większej pamięci niż dostępna w ZX Spectrum. Metodę tą wykorzystano także do zabezpieczania programów. Program „właściwy" zostaje zapamiętany na taśmie bez nagłówka (w gwarze stosuje się nazwę: „bajty bez nagłówka"). Jest on poprzedzany króciutkim programem, który ładuje blok bez nagłówka ustawiając najpierw parametry transmisji i wywołując procedurę systemową (sposób opisano na początku tego punktu). „Bajty bez nagłówka" można skopiować wykorzystując systemową procedurę ładowania.

Odmienny system zabezpieczania programu wykorzystuje zmienną systemową FEAMES o adresie 23672, która służy do zliczania czasu (patrz 2.2.7).

Jedna z nakładek ustawia w komórkach FRAMES pewną wartość, a program właściwy odczytuje je. W ten sposób program może „stwierdzić", czy odległość na taśmie pomiędzy nakładkami jest nie zmieniona, i gdy stwierdzi zmianę, zeruje pamięć. Metodą tą zabezpieczony jest przed skasowaniem np.: program *Ant Attack*. Program można skopiować, lecz kopia po wczytaniu rozpoczyna pracę, po czym zeruje pamięć. Jednym ze sposobów unieszkodliwienia tego zabezpieczenia jest dobieranie odstępu (na taśmie) między nakładkami metodą prób i błędów. Można także usunąć z programu głównego fragment, który sprawdza stan komórek FRAMES. Gdy blok główny jest napisany w Basicu, to jest to zadanie proste. Jeżeli jednak jest on napisany w kodzie wewnętrznym, jest to bardziej żmudne. Wymaga znajomości programowania w języku maszynowym oraz wykorzystania programu monitora-debuggera (patrz p. 8.10).

Innym sposobem uniemożliwiania skopiowania jest napisanie własnego programu ładującego (ang. *loader*) wykorzystującego inny od standartowego format zapisu informacji lub kodowania bitów na taśmie. Taki program ładujący jest zapamiętany jako pierwszy. Ładuje on pozostałą część programu.

Uwaga *Ogólnie rozpowszechnione programy kopiujące (np.: COPY COPY) pozwalają na „złamanie" większości opisanych w tym rozdziale zabezpieczeń.*

Można też zabezpieczyć program przed kopiowaniem za pomocą hasła (ang. *password*). Na przykład program *Blast Basic* sprzedawany jest razem z kartonikiem — tabelą złożoną z około 4000 kolorowych kwadracików. Tabela ta jest zapisana w programie. Program na początku swego działania pyta o kolor losowo wybranego pola tabeli. Podanie prawidłowych odpowiedzi umożliwia dalszą pracę programu. Perspektywa przerysowywania tabeli może zniechęcić do skopiowania programu.

Kosztowne programy są sprzedawane ze specjalnym kluczem elektronicznym, który podłącza się do szyny komputera. Program co pewien czas sprawdza czy klucz jest podłączony i kończy pracę, jeżeli stwierdzi jego brak. Jest to dosyć skuteczny sposób ochrony, bowiem można mieć kilka kopii programu, lecz tylko jedna z nich może być używana (o ile oczywiście ktoś nie skonstruuje elektronicznego wytrycha).

Na zakończenie podany zostanie przykład zabezpieczania, które Czytelnik może zastosować w swoim programie. Jest to zabezpieczenie „z hasłem". Do zabezpieczenia wykorzystuje się operacje różnicy symetrycznej (XOR). Operuje ona na dwu argumentach i daje w wyniku 0, gdy są one równe, i 1, gdy różnią się. Procesor wykonuje instrukcję obliczającą różnicę symetryczną dwu bajtów „bit po bicie". Do zabezpieczenia programu wykorzystano zależność, że

$$a \text{ XOR } b \text{ XOR } b = a$$

Program (w Basicu) należy przekształcić w ten sposób, że każdy bajt programu zostanie poddany operacji różnicy symetrycznej z pewnym wybranym przez Czytelnika bajtem — hasłem i w tej postaci zapisany na taśmie.

Program tak zapamiętany załadowany do pamięci startuje automatycznie i pyta o hasło. Następnie przekształca program w ten sposób, że wykonuje operacje XOR z hasłem podanym w odpowiedzi. Jeżeli hasło było poprawne, to treść programu zostaje odtworzona automatycznie. Gdy było błędne, w pamięci pozostają nic nie znaczące bajty.

```
9989 STOP
9990 GO SUB 9997: LET a=PEEK 236
35+256*PEEK (23636)+4: INPUT "Ha
slo (jedna litera) = ";a$: LET B
1=CODE (a$)
9991 LET b=PEEK a: IF b<>13 THEN
 GO TO 9994
9992 IF (PEEK (a+1)*256+PEEK (a+
2))=9990 THEN PRINT AT 0,16;" "
: STOP
9993 LET a=a+5: GO TO 9991
9994 PRINT FLASH 1;AT 0,16;"©":
POKE a,FN x(b,b1): LET a=a+1: GO
 TO 9991
9997 DATA 221,42,11,92,221,126,4
,221,174,12,79,6,0,201
9998 RESTORE : FOR A=23296 TO 23
309: READ B: POKE A,B: NEXT A: R
ETURN
9999 DEF FN x(Y,Z)=USR 23296
```

Podany wyżej program należy dołączyć na końcu programu, który ma zostać zabezpieczony. Wykorzystuje on asemblerową procedurę obliczającą różnicę symetryczną. Procedura ta umieszczana jest w buforze drukarki (linie 9997,9998). Wywołanie jej w instrukcji DEF FN pozwala na przekazanie parametrów. Procedura ta została opisana w rozdziale 4.4.

Linie o numerach 9991÷9994 przekodowują program. Wykonują one operacje różnicy symetrycznej na wszystkich bajtach programu, aż do linii o numerze 9990. W zakodowanym programie numery wszystkich linii pozostają niezmienione. Zabezpieczony program wraz z podanym wyżej fragmentem uruchamia się instrukcją:

GOTO 9990

Spowoduje to zakodowanie programu zgodnie z podanym w linii 9990 hasłem. Zakodowany program można zapamiętać:

SAVE "*nazwa*" LINE 9990

Załadowany z taśmy wystartuje automatycznie pytając o hasło. Program zostanie odkodowany, gdy hasło będzie poprawne. Można go wtedy uruchomić.

** Dźwięk 5

Jak wiadomo ZX Spectrum ma wbudowany głośnik. Za jego pośrednictwem można usłyszeć, że wczytuje się program, uzyskać potwierdzenie wciśnięcia klawisza. Język Basic ma także instrukcję, za pomocą której można generować dźwięk:

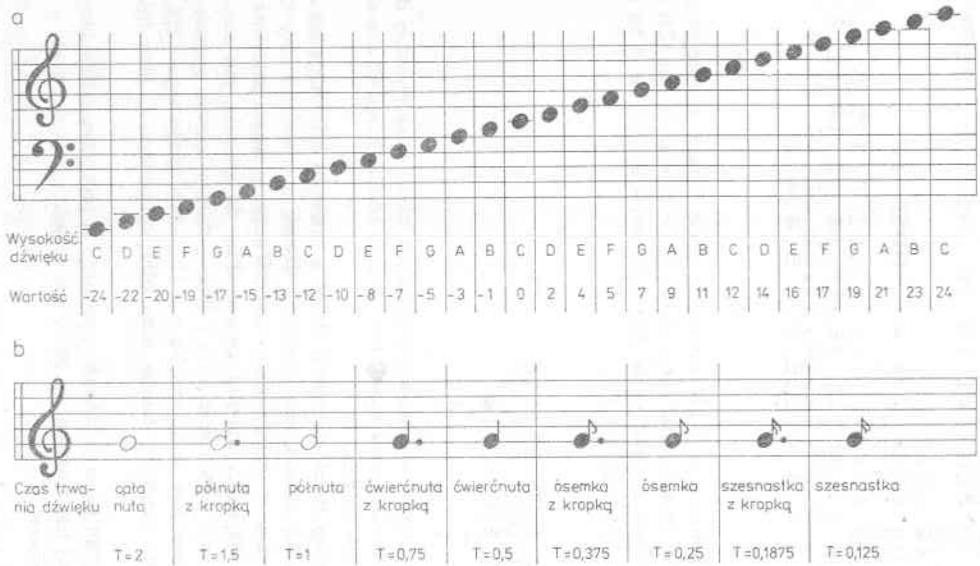
```
BEEP x, y
     ↑  ↑
        względna wysokość tonu (−60 do +69 półtonów
        względem środkowego C)
     długość trwania dźwięku w sekundach (od 0 do 10.499)
```

Jak tworzony jest dźwięk w ZX Spectrum? Sprzęt służący do tego celu jest bardzo prosty. Generator dźwięku związany jest bezpośrednio z układami obsługi magnetofonu. Czwarta linia szyny danych współpracujących z portem o numerze 254 jest dołączona do głośnika. Gdy wartość tego bitu wynosi 1 membrana głośnika unosi się, gdy 0 — opada. Generacja dźwięku polega na szybkim zmienianiu stanu logicznego tej linii. Na skutek drgań membrany wywołanych tymi zmianami powstaje dźwięk. Wysokość tonu zależy od częstotliwości drgań, jego długość od ich liczby. Ze względu na stałą amplitudę podanego sygnału nie ma możliwości zmian głośności.

Ten prymitywny sposób tworzenia dźwięków jest jednak dobrze oprogramowany. Procesor systemu może wyliczyć szybkość zmian linii D4 i w ten sposób reguluje wysokość dzwięku. Instrukcja BEEP uwzględnia skalę muzyczną.

Na rysunku 27 podano wartości parametrów określających wysokość tonu w instrukcji BEEP. Dla nut wyższych o pół tonu (z krzyżykiem #) należy dodać 1, a dla dźwięków niższych o półtonu (z bemolem b) odjąć 1. Na rys. 27b pokazano odmierzanie czasu trwania różnych dźwięków.

Poniższy program pozwala wysłuchać melodii „Romance d'amour" granej przez ZX Spectrum, w dowolnym określonym przez użytkownika tempie.



Rys. 27. Parametry instrukcji BEEP:
a — wysokość tonu, b — długość dźwięku

```
  5 PRINT AT 0,0;"tempo wolne=3
  6 PRINT "tempo umiarkowane=2"
  7 PRINT "tempo szybkie=1"
  8 INPUT "podej tepo? ";k
 20 LET k=k/8
 25 RESTORE
 30 FOR i=1 TO 45: READ a: BEEP
k,a
 40 NEXT i
 60 BEEP 2*k,4: PAUSE 10
 70 GO TO 25
100 DATA 11,11,11,11,9,7,7,6,4,
4,7,11,16,16,16,16,14,12,12,11,9
,9,11,12,11,12,11,15,12,11,11,9,
7,7,6,4,6,6,6,6,7,6,4,4,4
```

Jak już wspomniano port 254 kontroluje wiele funkcji wykorzystywanych przez system m.in. generację dźwięku (D4) i kolor brzegu ekranu (D0 i D2). Dźwięk można zatem uzyskać także aktywując bezpośrednio bit 4 tej szyny. Można to zrobić instrukcją OUT.

```
10 OUT 254,16
20 OUT 254,0
30 GO TO 10
```

Ze względu na powolność działania ZX Spectrum języka Basic powstały dźwięk przypomina raczej buczenie niż wysoki ton. Zmiana koloru brzegu ekranu wynika z podania D2 i D4 równego 0.

Procedury obsługi BEEP zawarte w ROM-ie zawierają informację jak często musi być pobudzany głośnik, aby otrzymać dźwięk o wysokości nuty C (dla ciekawych: membrana musi się poruszyć co 6689 okresów zegara procesora Z80).

Podprogram wysyłający impulsy bezpośrednio do głośnika jest sterowany dwoma parametrami: wysokością tonu f [Hz] i czasem trwania sygnału t [s]. W rejestrach DE znajduje się iloczyn f*t, a w HL liczba taktów zegara ZX Spectrum konieczna do generacji danego tonu (dla środkowego C zawartość HL wynosi 1642).

Poniższy program generuje dźwięk wywołując tę procedurę systemową. Długość pętli czasowej jest zmienna, co powoduje zmianę wysokości tonu.
Program asemblerowy:

```
LD DE,0205 — ton C o czasie trwania 1 s
LD HL,N    — wysokość dźwięku
CALL 03B5
RET
```

jest wykorzystany w tym programie:

```
   5 FOR k=0 TO 20
   9 RESTORE
  10 DATA 17,05,01,33,1+k*10,0,2
05,181,3,201
  20 FOR i=60000 TO 60009
  30 READ a: POKE i,a
  40 NEXT i: REM wczytano parame
try
  50 RANDOMIZE USR 60000: REM ge
neracja dzwieku
  60 NEXT k
```

Instrukcja BEEP pozwala tylko na generację tonów o określonej wysokości. Nie można zatem za jej pomocą tworzyć przebiegów szumowych. Przebieg taki można jednak utworzyć zmieniając w sposób losowy wysokość generowanego tonu.

Przedstawiony poniżej program, korzystający z podprogramu asemblerowego*), do generacji dźwięku używa losowo wybranych bajtów z pamięci stałej komputera i instrukcją OUT steruje pracą głośnika. W czasie pracy programu zmienia się kolor brzegu ekranu.

Adres	Program	Kod
60000	LD B,255	06,255
60002	LD HL,0	21,0,0
60005 A2	LD A,(HL)	126
60007	OR B	246,8
60009	OUT (254),A	211,254
60011	LD C,128	14,128
60012 A1	DEC C	13
	JP NZ,A1	194,108,234
	INC HL	35
	DEC B	5
	JP NZ,A2	194,101,234
	RET	201

Uruchomienie tego programu

```
   9 RESTORE
  10 DATA 6,255,21,0,0,126,246,8
,211,254,14,128,13,194,108,234,3
5,5,194,101,234,201
  20 FOR i=60000 TO 60021
  30 READ a: POKE i,a
  40 NEXT i: REM wczytano parame
try
  50 RANDOMIZE USR 60000: GO TO
50: REM generacja dzwieku
  60 GO TO 50
```

pozwoli uzyskać przybliżony dźwiękowy szumowy.

*) Program asemblerowy powinien być umieszczony powyżej komórki o adresie 32767. W przeciwnym razie program ten będzie zatrzymywany (co 1/50 s) przez procedurę obsługi przerwań z układu ULA. W efekcie generowany dźwięk będzie zniekształcony.

Warto pamiętać, że w trakcie pracy BEEP zablokowane jest przyjmowanie przerwań. Zawartość licznika FRAMES nie jest zmieniana i nie jest obsługiwana klawiatura, a co za tym idzie nie jest możliwe przerwanie generacji dźwięku (także klawiszem BREAK). Procesor jest bowiem stale zaangażowany w jego tworzenie, nie może więc wykonywać obliczeń związanych z programem.

Znacznie lepsze efekty dźwiękowe można uzyskać dołączając do głośnika wewnętrznego mikrokomputera dodatkowy układ wzmacniający. Sygnał podawany na głośnik ze względu na wspólne połączenia dociera także do gniazd EAR i MIC. Poziom tego sygnału odpowiada poziomom wejść uniwersalnych lub magnetofonowych wzmacniaczy.

Choć sposób powstawania dźwięku jest bardzo prosty, to dzięki odpowiedniemu sterowaniu głośnika twórcy programów komercyjnych potrafią uzyskiwać interesujące efekty (np. *Jet Set Willy*, *Maniac Miner*). Mało tego, niektóre z tych programów potrafią syntezować mowę, co prawda mało zrozumiałą (np. *Opening James Bond*).

Podłączając przez szynę wyjściową komputera dodatkowy głośnik można utworzyć własny syntezer dźwięku. Może być on np. sterowany zamianami z jednej z linii danych. W ten sposób, odpowiednio oprogramowując w języku wewnętrznym operacje tworzenia dźwięku, można na bazie ZX Spectrum konstruować rozmaite syntezatory muzyczne, nawet polifoniczne.

*** Układ Interface 1 oraz microdrive 6

* Możliwości układu Interface 1 6.1

Układ Interface 1 z microdrive'm to efektywna pamięć masowa komputera ZX Spectrum, która może być używana zamiast magnetofonu. Interface 1 pozwala podłączyć do komputera od 1 do 8 stacji microdrive'ów. Zwiększa on także możliwości programowe ZX Spectrum przez rozszerzenie języka Basic. Tylko niektóre nowe rozkazy dotyczą microdrive'ów.

Dołączenie Interface 1 umożliwia łączenie komputerów ZX Spectrum w sieć lokalną. Komputery te mogą przesyłać informacje między sobą. Słowo „lokalna" oznacza, że komputery znajdują się w tym samym pomieszczeniu lub budynku. W sieci lokalnej kilka komputerów (w tym przypadku do 64 sztuk) wraz z urządzeniami zewnętrznymi jest połączonych ze sobą specjalnym łączem znajdującym się w układzie Interface 1.

Sieć komputerowa stwarza możliwości:
— przesyłania informacji i programów między komputerami,
— zbierania danych z kilku komputerów,
— używania przez kilka komputerów urządzeń zewnętrznych, podłączonych do jednego z nich,
— dostęp do wspólnego banku danych przez wiele maszyn itd.

Interface 1 pozwala podłączyć bezpośrednio do Spectrum urządzenie zewnętrzne wyposażone w standardowe łącze szeregowe RS 232, tj. takie, w którym dane przesyłane są bit po bicie (patrz p. 6.4).

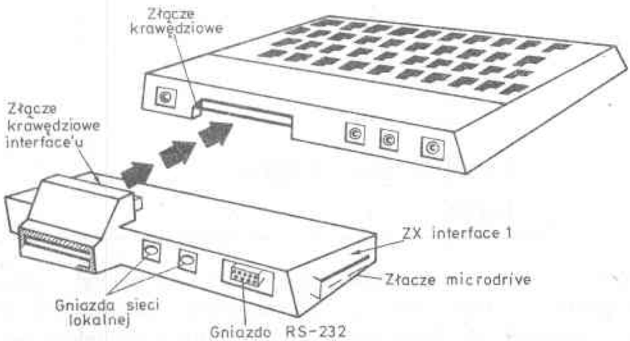
Dołączenie Interface 1 umożliwia rozszerzenie mechanizmu strumieni danych i kanałów. W „rozszerzonym" ZX Spectrum dostępne są standardowo nowe kanały:

„m" — microdrive,
„t" — kanał realizujący przesyłanie tekstu w kodzie ASCII, w łączu o standardzie RS 232,

„b" — kanał umożliwiający przesyłanie dowolnych danych przez łącze RS 232,
„n" — kanał do transmisji informacji w sieci lokalnej.

Układ Interface 1 jest dość skomplikowany. Wewnątrz znajduje się dodatkowa pamięć stała (ROM), ULA oraz wiele tranzystorów i oporników. W pamięci stałej zapisano programy rozszerzające możliwości systemu operacyjnego ZX Spectrum.

Interface 1 podłącza się do szyny krawędziowej z tyłu ZX Spectrum (ang. *edge connector*) — rys. 28.



Rys. 28. Sposób połączenia Interface 1 z ZX Spectrum

Interface 1 ma czternastoprzewodową szynę służącą do podłączenia pierwszego microdrive'u (każdy kolejny microdrive podłącza się do poprzedniego), gniazdo RS 232, dwa gniazdka służące do łączenia komputerów w sieć oraz przedłużenie szyny krawędziowej ZX Spectrum.

* * Dodatkowe instrukcje języka Basic 6.2

Rozszerzenie instrukcji SAVE, LOAD, MERGE, VERIFY 6.2.1

Komputer wyposażony w układ Interface 1 może przesyłać programy za pomocą LOAD, SAVE i in. (podobnie jak odbywa się to z magnetofonem) do:
— microdrive'u,
— innego komputera włączonego do sieci,
— przez łącze RS 232 do innych urządzeń zewnętrznych,
Na przykład, wykonanie instrukcji:

SAVE * "m" ; 1 ; "*nazwa*"

nazwa programu
numer stacji microdrive (1÷8
litera „m" oznacza, że instrukcja dotyczy microdrive
znak * informuje o rozszerzeniu instrukcji

spowoduje zapamiętanie programu o podanej nazwie na kasetce microdrive'u. Podobną postać mają inne instrukcje, np.:

MERGE * "m"; 1; "*nazwa*"

Rozszerzonych instrukcji można używać ze wszystkimi opcjami, jakich używa się podczas współpracy z magnetofonem (patrz rozdz. 1.5), np.:

SAVE * "m"; 2; "*nazwa*" LINE *numer linii*
SAVE * "m"; 3; "*nazwa*" DATA *nazwa tablicy*
SAVE * "m"; 5; "*nazwa*" CODE *adres, długość*
SAVE * "m"; 8; "*nazwa*" SCREEN$

Analogicznie: VERIFY *, LOAD *, MERGE *.

Uwaga *Zamiast nazwy nie można podać "" (nazwy pustej).* *Niekiedy zachodzi potrzeba zapisania na kasetce kilku kopii tego samego programu, w tym celu należy w komórce o adresie 23791 wpisać żądaną liczbę kopii (POKE 23791, liczba) oraz wykonać SAVE *.*

Wszystkich wyżej wymienionych możliwości można używać także podczas przesyłania programów w sieci lokalnej lub za pomocą łącza RS 232.

Na kasetce microdrive'u można zapisać samostartujący program za pomocą SAVE LINE, podobnie jak na taśmie magnetofonowej. Szczególnie często używany program może być zapamiętany na kasetce microdrive'u pod nazwą „run". Program taki wpisuje się do pamięci komputera i uruchamia za pomocą instrukcji RUN (ENTER), bez potrzeby pisania LOAD. Opisany mechanizm zadziała, jeśli zostaną spełnione warunki:

— program ma, jak powiedziano, nazwę „run",
— kasetka zostanie umieszczona w microdrive'ie nr 1 (tzn. pierwszym dołączonym do Interface 1),
— instrukcja RUN wykonana będzie bezpośrednio po włączeniu komputera lub po instrukcji NEW.

Do wczytania programów przesyłanych w sieci lokalnej używa się instrukcji o postaci: LOAD* "n"; *nr*. Komputer, który wysyła program powinien wykonać instrukcję np.:

SAVE* "n" ; 3

numer odbiorcy programu — liczba od 1 do 64 (każdy komputer pracujący w sieci ma własny numer)
literka „n" oznacza, że operacja dotyczy sieci lokalnej

Uwaga *Nadawca (tzn. komputer wykonujący SAVE) nie może zasygnalizować odbiorcy, że zamierza przesłać program lub dane. Odbiorca powinien wiedzieć skądinąd, że powinien wykonać instrukcję LOAD.*

Komputer nadający program oczekuje na zwolnienie sieci oraz na potwierdzenie od odbiorcy gotowości do transmisji. W tym czasie kolor brzegu ekranu nadawcy zmienia się na czarny. Gdy rozpocznie się właściwa transmisja brzeg ekranu odbiorcy i nadawcy błyska (identycznie w obu komputerach).

Przesłany w sieci program można zweryfikować. Odbiorca winien wykonać:

VERIFY* "n"; *nr nadawcy*

Nadawca powtarza instrukcję:

SAVE* "n"; *nr odbiorcy*

Możliwe jest także przesłanie programu do wszystkich komputerów włączonych w sieć (określone terminem ang. *broadcasting*) za pomocą:

SAVE* "n"; 0

↑ liczba 0 jako numer odbiorcy (nadawcy) oznacza, że transmisja dotyczy wszystkich komputerów

Komputery, które chcą odebrać program powinny wykonać:

LOAD* „n" ; 0

W takim przypadku nadawca nie oczekuje potwierdzenia od odbiorców i od razu rozpoczyna transmisję. Może więc zdarzyć się, że żaden z komputerów nie odbierze programu.

Podobnie jak w sieci, przesyła się programy za pomocą łącza RS 232. Najczęściej transmituje się programy przez zwykłą linię telefoniczną za pośrednictwem specjalnych urządzeń, tzw. modemów. Do transmisji programu można użyć tylko kanału „b" (z łączem RS 232 współpracuje także kanał „t" patrz rozdz. 6.4). Nadawca powinien wykonać instrukcję:

SAVE* "b"

Jednocześnie odbiorca:

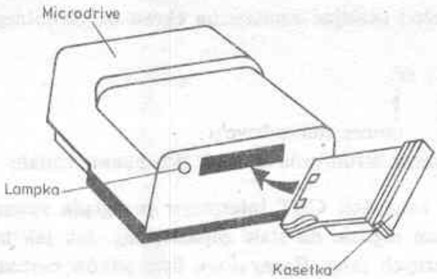
LOAD* "b"

Łącze RS 232 używane w układzie Interface 1 ma programowaną szybkość transmisji. Zarówno nadawca, jak i odbiorca powinien pracować z tą samą szybkością (sposoby jej zmiany podano w p. 6.2.5).

Instrukcje obsługi microdrive 6.2.2

Microdrive zapisuje dane na specjalnej kasetce. Zawiera ona pętlę cieniutkiej taśmy magnetycznej (znacznie cieńszej i węższej od magnetofonowej) (rys. 29). Informacje zapisywane są na taśmie w postaci bloków o stałej długości zwanych sektorami. Organizacja kasetki została opisana w p. 6.3.

Na kasetce można zapisywać programy (za pomocą SAVE) oraz dane. Ciąg



Rys. 29. Microdrive

informacji na kasetce (np. program) nosi nazwę pliku (ang. *file*). Plik może zajmować wiele sektorów rozrzuconych po całej kasetce. W poprzedniej części rozdziału opisano sposób tworzenia plików zawierających program (SAVE*). W następnej zostaną opisane zasady tworzenia i odczytywania plików danych. Niżej zaś podano inne instrukcje obsługi microdrive'u.

Nową kasetkę należy przed użyciem przygotować do pracy. Instrukcja FORMAT nadaje kasetce nazwę oraz dzieli ją na sektory. Jeżeli kasetka była używana, stare informacje są kasowane. W czasie formatowania kasetka jest testowana. Jeżeli pewne fragmenty tasiemki są uszkodzone lub sklejone, sektory, w których znajdują się wadliwe fragmenty, oznaczane są jako „zajęte" i nie mogą być użyte. Instrukcja ta ma następującą postać:

FORMAT "m"; *nr*; "*nazwa*"

- "*nazwa*" — nazwa kasetki
- *nr* — numer microdrive'u (1 ÷ 8)
- "m" — literka „m" oznacza, że instrukcja dotyczy microdrive'u (działanie tej instrukcji jest zupełnie inne, jeśli odnosi się do sieci lub łącza RS 232 — patrz p. 6.4 i 6.5)

Wykonanie tej instrukcji trwa około 20 s. W tym czasie zmienia się kolor brzegu ekranu. Gdy formatuje się zupełnie nową kasetkę, to warto instrukcję FORMAT wykonać dwa razy z rzędu. Należy sprawdzić czy kasetka została dokładnie włożona do microdrive. W przeciwnym razie źle przylegająca do głowicy taśma zostanie uznana za uszkodzoną.

Polecenie:

CAT *nr*

- *nr* — numer microdrive'u

powoduje wyświetlenie na ekranie telewizyjnym nazw wszystkich plików znajdujących się na kasetce w microdrive'ie o podanym numerze. Najpierw drukowana jest nazwa kasetki, następnie tekst pusty (tzn. znak CR — zmiana wiersza), wreszcie nazwy wszystkich plików oraz ilość wolnej pamięci (w kilobajtach).

Katalog plików można przesłać zamiast na ekran do dowolnego strumienia danych:

CAT#K, *nr*

↑ numer microdrive'u
↑ numer strumienia danych lub nazwa kanału

Podczas wykonywania instrukcji CAT interpreter przegląda zawartość kasetki. Katalog nie jest bowiem nigdzie na stałe zapamiętany, tak jak ma to miejsce np. na dyskach elastycznych (ang. *floppy disc*). Spis plików można zapamiętać, np. w dodatkowym pliku:

```
   5 INPUT "Podaj numer microdri
ve'u ";nr
  10 OPEN #15;"m";nr;"katalog"
  20 CAT #15,nr
  30 CLOSE #15
```

Plik zapisany na taśmie można w prosty sposób zabezpieczyć, tak by jego nazwa nie pojawiła się na wydruku (lub w pliku) utworzonym instrukcją CAT. Powinien on mieć nazwę rozpoczynającą się od znaku o kodzie 0. Plik taki można utworzyć instrukcją:

SAVE * "m"; 1; CHR$ (0) + "*nazwa*"

lub

OPEN #4;"m"; 1; CHR$(0) + "*nazwa*"

Uwaga *Instrukcja CAT wykonana bezpośrednio po poleceniu FORMAT informuje ile wolnej pamięci znajduje się na kasetce (w kB).*

Niepotrzebny plik usuwa się, zwalniając tym samym miejsce na kasetce, za pomocą instrukcji:

ERASE "m"; *nr*; "*nazwa pliku*"

↑ numer microdrive'u

Rozszerzenie mechanizmu strumieni i kanałów 6.2.3

Dołączenie Interface 1 rozszerza zbiór standardowych kanałów (tzn. zbiór: „k", „p", „s") dostępnych w ZX Spectrum o nowe kanały:

„m" — microdrive,
„b", „t" — RS 232,
„n" — sieć lokalna.

Kanał otwiera się (łączy ze strumieniem danych) instrukcją OPEN. Zapisu i odczytu danych dokonuje się za pomocą instrukcji PRINT, INPUT, INKEY$, LIST — podobnie jak w czasie współpracy z kanałami: „k", „s" i „p" — patrz rozdz. 4. Język Basic został rozszerzony o nową instrukcję MOVE przysyłającą dane z jednego strumienia (kanału) do drugiego.

Kanał microdrive'u 6.2.4

Na kasetce microdrive'u można zapisać pliki dwu typów:

— programy, tablice lub bloki bajtów zapisane instrukcją SAVE*; pliki takie mogą być odczytane jedynie za pomocą LOAD* lub MERGE*,

— informacje zapisane za pomocą PRINT lub LIST; mogą one być odczytane przy użyciu INPUT oraz INKEY$.

Otwarcie strumienia danych dla kanału „m" powoduje utworzenie na kasetce pliku i nadanie mu nazwy. Postać instrukcji otwarcia pliku jest następująca:

OPEN#5; "m"; 1; "*nazwa*"

- "*nazwa*" — nazwa pliku na kasetce
- 1 — numer microdrive'u
- "m" — „m" oznacza, że operacja dotyczy microdrive'u
- 5 — numer strumienia

Na kasetce założony zostaje plik sekwencyjny, tj. taki, w którym informacje mogą być zapisywane jedna po drugiej.

Wpisywanie informacji

Wykonuje się je za pomocą instrukcji PRINT lub LIST (identycznie jak do dowolnego strumienia danych — patrz rozdz. 4.6) np.:

```
10 OPEN #5;"m";1;"dane"
20 FOR i=1 TO 50
30 PRINT #5;RND
40 NEXT i
50 CLOSE #5
```

Podany wyżej program zapisuje w pliku na kasetce microdrive nr 1 50 losowo wybranych liczb.

Wykonanie instrukcji PRINT nie zawsze powoduje zapisanie informacji na kasetce. W czasie otwierania pliku w pamięci ZX Spectrum w obszarze CHANS (patrz rozdz. 4.6) jest umieszczany bufor pozwalający na zapamiętanie 512 bajtów. PRINT przesyła dane do bufora. Są one kopiowane na kasetkę, gdy bufor jest pełen lub gdy wykonana zostanie instrukcja CLOSE#.

Treść programu można przesłać do pliku za pomocą instrukcji LIST. Plik taki może być odczytany, np. znak po znaku, za pomocą INKEY$ lub INPUT. Nie może być natomiast wczytany instrukcją LOAD.

Jeśli pojedyncza instrukcja PRINT zapisuje kilka liczb, a tak zapisany plik ma być odczytany za pomocą INPUT (jako ciąg liczb), to praktycznie jedynymi separatorami, jakie mogą być użyte w instrukcji PRINT (separatorami oddzielającymi od siebie drukowane liczby), mogą być „;" oraz „'" (apostrof). Dane są umieszczane w pliku, podobnie jak na ekranie, jako ciąg znaków. Na zakończenie wykonania instrukcji PRINT jest wysyłany znak o kodzie 13 (CR). Użycie np. przecinka jako separatora między liczbami w instrukcji PRINT powoduje zapamiętanie ciągu spacji (znaków odstępu). Przy odczycie, gdy w instrukcji INPUT wczytana ma być liczba, a na taśmie liczbę tę poprzedzają znaki spacji, to wystąpi błąd.

Pliki utworzone instrukcją PRINT można odczytywać także za pomocą funkcji INKEY$. Funkcja ta zawsze daje w wyniku znak (inaczej niż przy współpracy z klawiaturą, siecią lokalną lub łączem RS 232, gdzie wynikiem może być tekst pusty).

Odczytywanie informacji

Informacje zapisane przez podany wyżej program można odczytać np. w taki sposób:

```
 5 DIM a(50)
10 OPEN #5;"m";1;"dane"
20 FOR i=1 TO 50
30 INPUT #5;a(i): PRINT a(i)
40 NEXT i
50 CLOSE #5
```

W czasie wykonania instrukcji INPUT po raz pierwszy, z kasetki odczytuje się pełen bufor danych (512 bajtów). Informacje są następnie pobierane z bufora aż do jego wyczerpania. Wtedy wczytuje się do bufora następny sektor, itd.

Gdy otwiera się plik (OPEN #) interpreter sprawdza, czy na kasetce znajduje się plik o podanej nazwie. Jeśli plik istnieje, to zostaje on otwarty „dla odczytu" — co oznacza, że nie można do niego niczego zapisywać. Jeśli pliku na kasetce nie ma, otwiera go się „dla zapisu", a więc można do takiego pliku tylko pisać. Gdyby linia nr 30 w powyższym programie miała postać:

30 INPUT #5, A(I)

tzn. zamiast średnika wpisano by przecinek, to interpreter nie zasygnalizowałby błędu przy wpisywaniu programu (instrukcja tu jest gramatycznie poprawna w myśl reguł Basica), lecz w czasie wykonania programu wystąpiłby błąd „próba wpisania do pliku otwartego dla odczytu". Bowiem instrukcja INPUT (z przecinkiem) przed wczytywaniem informacji przesyła do pliku ciąg spacji! Niewprawny programista może dosyć łatwo przeoczyć tak subtelny błąd.

W ZX Spectrum nie można dopisywać informacji do istniejącego pliku. Jedynym sposobem rozszerzania pliku jest przepisanie go do nowego pliku, a następnie zapisanie dalszych informacji, podobnie jak zrobiono w programie:

```
  10 OPEN #6;"m";1;"dane1"
  20 OPEN #5;"m";1;"dane"
  30 FOR i=1 TO 50
  40 INPUT #5;a: PRINT #6;a
  50 NEXT i: REM Przepisanie pli
ku "dane" do pliku pomocniczego
"dane1"
  60 CLOSE #5: ERASE "m";1;"dane
"
  70 FOR i=1 TO 10
  80 PRINT #6;RND
  90 NEXT i: REM Dopisanie infor
macji
 100 CLOSE #6
```

Fragment programu z linii od 10 do 50 realizuje przepisanie jednego pliku do drugiego. Czynność taką można zrealizować za pomocą pojedynczej instrukcji MOVE (instrukcja ta zastępuje linie 30, 40, 50):

30 MOVE #5 TO #6

↑ źródło danych (#5)   ↑ miejsce przeznaczenia (#6)

Umożliwia ona przesyłanie danych z dowolnego strumienia do innego np.:

10 MOVE #1 TO#2

powoduje, że informacje wczytane z klawiatury będą wyświetlane na ekranie.

Uwaga — *Wykonanie tej instrukcji można zatrzymać jedynie wtedy, gdy na ekranie pojawi się napis: scroll?, a w odpowiedzi wciśnięty zostanie klawisz BREAK, N lub SPACE.*

W instrukcji MOVE zamiast nazwy strumienia można podać specyfikację kanału (pliku, kanału sieci lokalnej), np. MOVE "n"; 4 TO#5 itd. Poprawna jest instrukcja MOVE w jednej z poniższych postaci, np.:

MOVE "m"; 1; "*dane*" TO #6

lub MOVE "m"; 1; "*dane*" TO "m"; 1; "*dane 1*"

albo MOVE #5 TO "m"; 1; "*dane 1*"

Jeśli podana zostanie specyfikacja pliku np. "m"; 1; "*nazwa*", to interpreter automatycznie otwiera go na początku wykonywania instrukcji MOVE, a zamyka po jej zakończeniu. Gdy natomiast poda się numer strumienia, to musi on być wcześnie otwarty, a po zakończeniu operacji pozostaje nadal otwarty.

Program dopisujący informacje do pliku może być zmodyfikowany np. w ten sposób:

```
10 OPEN #6;"m";1;"dane1"
20 MOVE "m";1;"dane" TO #6
30 FOR i=1 TO 10
40 PRINT #6;RND
50 NEXT i
60 CLOSE #6
```

W programie napisanym w języku Basic ZX Spectrum nie ma standardowo możliwości wykrycia w czasie odczytu końca pliku. W programach podanych wyżej ilość zapisanych informacji (liczb) była znana z góry. Jeśli jednak nie jest znana? Można sprawdzać (wykorzystując wiadomości z rozdz. 6.3) czy w buforze są jeszcze informacje. Metoda ta zawodzi, gdy plik ma długość równą wielokrotności liczby 512. Innym sposobem jest zapisywanie na końcu pliku nietypowych danych, np.: dwu znaków o kodzie 0. Plik taki niezależnie od tego, czy zawiera łańcuch znaków czy ciąg liczb powinien być odczytywany znakowo.

Oto przykład wypełniania i czytania pliku liczb:

```
  10 OPEN #5;"m";1;"dane2"
  20 INPUT "Podaj ilosc liczb ";
nr
  30 FOR i=1 TO nr
  40 PRINT #5;RND
  50 NEXT i: REM Wypelnianie pli
ku
  60 PRINT #5;CHR$ (0);CHR$ (0):
 REM Zapisanie znacznika konca
 pliku
  70 CLOSE #5
  80 REM Odczytywanie
  90 OPEN #5;"m";1;"dane2"
 100 INPUT #5;a$
 110 IF a$=CHR$ (0)+CHR$ (0) THE
N GO TO 150
 120 PRINT VAL (a$)
 130 GO TO 100
 150 PRINT "koniec pliku": CLOSE
 #5
```

Jeżeli należy odczytać plik o nieznanej długości, który został uprzednio zapisany bez znaczników końca, to należy go poprawić (załóżmy, że plik ten ma nazwę „dane"):

```
  10 CLOSE #10: ERASE "m";1;"pom
ocniczy"
  20 OPEN #10;"m";1;"pomocniczy"
  30 MOVE "m";1;"dane" TO #10
  40 PRINT #10;CHR$ (0);CHR$ (0)
  50 CLOSE #10: REM Przepisanie
pliku i doppisanie znacznika kon
ca
  60 REM Teraz nastapi zmaina na
zwy
  70 ERASE "m";1;"dane"
  80 MOVE "m";1;"pomocniczy" TO
"m";1;"dane"
  90 ERASE "m";1;"pomocniczy"
```

Program kopiuje plik ,,dane" do pliku ,,pomocniczy" dopisując na koniec znacznik końca pliku.

W Basicu ZX Spectrum nie ma możliwości zmiany nazwy pliku. Jedynym rozwiązaniem jest usunięcie pliku ,,dane", skopiowanie pliku ,,pomocniczy" do nowego pliku ,,dane" oraz skasowanie pliku ,,pomocniczy". W wyniku opisanego wyżej ciągu czynności na kasetce pozostaje plik o nazwie ,,dane" z dopisanym znacznikiem końca pliku.

Program byłby całkowicie zrozumiały, gdyby nie linia nr 10. Można byłoby obejść się bez niej, lecz niekiedy dołożenie takiej ,,niepotrzebnej" z pozoru linii daje korzyści — chroni przed błędem. Często zdarza się, że plik pozostaje otwarty po zakończeniu wykonywania programu. Próba otwarcia otwartego pliku kończy się błędem. Jeśli jednak plik był zamknięty i wykona się instrukcję CLOSE, to nie zmienia ona niczego i program działa dalej. Wygodnie jest więc na początku programu zamknąć, na wszelki wypadek, plik (lub strumień), którego chcemy użyć. Podobnie rzecz się ma z plikami zapisanymi na kasetce. Często na kasetce pozostają stare, zapomniane pliki — najczęściej tymczasowe, pomocnicze. Programiści często nadają takim plikom podobne nazywy — imię sympatii lub ,,dane", ,,pomoc", itp. Gdy plik znajduje się na kasetce, to instrukcja OPEN otwiera go dla odczytu i próba zapisania czegokolwiek kończy się błędem. Jeśli plik nie istnieje, to instrukcja ERASE nie robi nic. Można więc śmiało użyć jej na początku programu, tak jak zrobiono to w linii 10.

Kłopoty z ,,kolorami"

Interpreter Basica zawiera następującą niekonsekwencję (nie jedyną zresztą). Jeżeli w programie, który zapisuje informacje do pewnego strumienia, użyje się instrukcji zmieniających atrybuty ekranu (np. kolor, tzn. PAPER, INK, BRIGHT itd.), to kody tych instrukcji zostaną przesłane do ostatnio używanego strumienia, a nie do ekranu — zmiana atrybutów ekranu nie nastąpi.

Gdy wykona się program:

```
 5 PAPER 7: CLS
10 OPEN #10;"m";1;"test"
20 PRINT #10;"dowolny tekst"
30 PAPER 4: CLS
40 PRINT #10;" inny tekst"
50 CLOSE #10
```

kolor tła ekranu nie zmieni się na zielony po wykonaniu linii 30, jak można by oczekiwać. Zamiast tego kody instrukcji PAPER oraz CLS znajdą się w pliku ,,test". Można się o tym przekonać wykonując instrukcję:

MOVE "m"; 1; ,,test" TO #2

Błąd ten nie wystąpi, gdy przed wykonaniem instrukcji zmieniającej atrybuty ekranu wydrukuje się na ekranie tekst pusty np.:

```
25 PRINT "";
```

Instrukcja ta spowoduje zmianę aktualnego kanału. Będzie nim teraz kanał ekranu.

Kanał RS 232 — 6.2.5

W ZX Spectrum rozszerzonym o układ Interface 1 można używać dwu kanałów, które transmitują informacje przez łącze w standardzie RS 232. Za pomocą tego łącza można do Spectrum podłączyć np. drukarkę lub modem wyposażone w standardowy układ sprzęgający RS 232 (dalsze informacje o RS 232 zawarto w p. 6.4).

Łącze RS 232 zawarte w Interface 1 ma zmienną, programowaną szybkość transmisji, objętą także standardem. Określa się ją w bitach na sekundę czyli w bodach.

ZX Spectrum przesyła informacje z szybkością wybraną spośród:

50, 110, 300, 600, 1200, 2400, 4800, 9600, 19800 bitów/s.

Szybkość transmisji należy zaprogramować zanim łącze RS 232 zostanie użyte. W tym celu można się posłużyć instrukcją:

10 FORMAT "*k*"; *szybkość*

↑ szybkość transmisji (jedna z podanych liczb)

↑ kanał: „t" lub „b" — obojętnie. W obu przypadkach jest ustawiana zawartość tej samej zmiennej systemowej

Strumień dla kanału „b" i „t" można otworzyć za pomocą instrukcji:

15 OPEN#10; "t"

↑ kanał: „t" lub „b"

Oba mogą być otwarte jednocześnie, lecz możliwe jest otwarcie tylko jednego strumienia dla każdego z nich. W obu wypadkach dane przesyłane są przez to samo łącze.

Kanały „b" i „t" różnią się sposobem traktowania informacji. Przez kanał „t" — tekstowy, można transmitować tylko znaki, które mogą być wydrukowane na drukarce (lub ekranie telewizyjnym). Jest on przeznaczony np. do drukowania treści programów.

W czasie pisania do kanału „t" bajty informacji traktowane są w sposób podany w tabl. 4.

W czasie czytania z kanału „t" ignorowane są najbardziej znaczące bity poszczególnych bajtów (tzn. wczytywane są bajty o wartościach 0÷127).

Tablica 4. Sposób interpretacji bajtu podczas transmisji przez kanał „t”

Wartość bajtu	Interpretacja
0÷31	znaki sterujące są pomijane. Wyjątkiem jest znak o kodzie 13 tzn. CR, który jest zamieniany na znaki CR, LF (13, 10), co oznacza zmianę wiersza drukarki
32÷127	znaki, które można bezpośrednio wydrukować (wg kodu ASCII): litery duże i małe, cyfry, separatory, operatory arytmetyczne są przesyłane bez zmian
128÷164	znaki graficzne (tzn. dostępne z klawiatury w trybie G), są pomijane. Zamiast nich drukuje się znak zapytania (?)
105÷255	kody słów kluczowych są zamieniane na ciągi znaków, np.: kod PRINT (jeden bajt) jest zastępowany znakami P, R, I, N, T

Kanał „b” przesyła wszystkie informacje bez żadnych zmian. Dobrze nadaje się więc do transmitowania programów za pomocą SAVE, np. przez linię telefoniczną.

Otwarcie kanału „t” i „b” jednocześnie jest często wygodne. Kanał „t” służy do przesyłania „listingów” (wydruków) programów, „b” — ich treści (o postaci używanej przez interpreter, patrz 2.5). Liczby lub teksty mogą być przesyłane zarówno przez kanał „t”, jak i „b” (instrukcje: PRINT, INPUT, INKEY$). Dobrze jest przydzielić kanał „t” do strumienia nr 3. Można wtedy używać instrukcji LPRINT i LLIST, które są równoważne: PRINT #3 i LIST #3. Zilustrowano to w poniższym programie:

```
   10 FORMAT "t";4800
   20 OPEN #3;"t"
   30 OPEN #4;"b"
   40 LPRINT "Ala ma kota": REM t
ransmisja do kanalu "t"
   50 LLIST
   60 PRINT #4;CHR$ (0);"dowolny
tekst";CHR$ (255): REM -do kanal
u "t"
   70 SAVE *"b"
   80 CLOSE #3: CLOSE #4
```

Uwaga — *Zamknięcie jednego z kanałów współpracujących z łączem RS 232 powoduje transmisję znaku zmiany wiersza (znak CR, o kodzie 13) po to, by opróżnić bufor drukarki (domniemywa się, że łącze było używane do współpracy z drukarką).*

Jeśli urządzenie, do którego Spectrum zamierza transmitować nie jest gotowe do współpracy, to kolor brzegu ekranu zmienia się na czarny, a Spectrum oczekuje na gotowość odbiorcy. W czasie transmisji na czarnym tle

ekranu są widoczne cienkie białe paski. Transmisję można przerwać za pomocą klawisza BREAK.

W zbiorze instrukcji języka Basic ZX Spectrum znajduje się instrukcja COPY. Powoduje ona skopiowanie obrazu wyświetlonego na ekranie na drukarkę. Za pomocą COPY można wydrukować obraz jedynie na drukarce podłączonej bezpośrednio na szynę krawędziową (ang. *edge connector*) z tyłu Spectrum (mogą to być drukarki ZX Printer lub Seikosha GP 50 A). W przypadku drukarki dołączonej do łącza RS 232 np.: Seikosha GP100 lub D100 można to jednak uczynić wykorzystując np. program (podany niżej), który kopiuje z ekranu tylko znaki zdefiniowane w generatorze znaków, a pozostałe zamienia na spacje:

```
  10 FORMAT "t";1200: REM Nalezy
 podac szybkosc transmisji odpow
iednia dla drukarki
  20 OPEN #3;"t"
  30 FOR x=0 TO 21
  40 FOR y=0 TO 31
  50 LET a$=SCREEN$ (x,y)
  60 LPRINT a$;
  70 IF a$="" THEN LPRINT " ";
  80 NEXT y
  90 NEXT x
  95 PRINT x,y
 100 CLOSE #3
 110 RETURN
```

Procedura przegląda ekran znak po znaku za pomocą funkcji SCREEN$ i odczytuje znaki z kolejnych miejsc ekranu (linia 50). Jeśli znak wyświetlony na ekranie nie jest określony w generatorze znaków, to funkcja SCREEN$ daje w wydruku tekst pusty, a program drukuje znak spacji (linie 60 i 70). Znaki graficzne są zamieniane na znak: ,,?"

Skopiowanie dowolnego rysunku z ekranu punkt po punkcie jest trudniejsze. Drukarka musi mieć możliwość pracy w trybie graficznym. Program zmienia się zależnie od typu drukarki. Drukarka kopiuje obraz wyświetlony na ekranie drukując rządki po 8 lub 7 (zależnie od typu drukarki) punktów w pionie 256 razy w wierszu. ,,Przestawienie" drukarki na taki tryb pracy wymaga przesłania kilku znaków sterujących, np.: dla drukarki Seikosha GP100 należy wysłać: LPRINT CHR$ (18). Informacje o kolejnych rządkach, które powinny być wydrukowane odczytuje się albo bezpośrednio z pamięci ekranu, albo za pomocą funkcji POINT (x, y) która daje w wyniku jeden, jeśli punkt ekranu o podanych współrzędnych ma kolor atramentu, a zero — gdy kolor tła.

Kanał sieci lokalnej **6.2.6**

Każdy komputer podłączony do sieci lokalnej ma własny numer z zakresu 1÷64. Numer nadaje się wykonując instrukcję:

10 FORMAT "n": *numer*

Można to także uczynić za pomocą:

10 POKE 23749, *numer*

Najczęściej sieć komputerowa jest używana do przesyłania programów. W sieci przesyłać można także:

— dane — nadawane za pomocą PRINT, a odbierane przy użyciu INPUT albo INKEY$ (funkcja INKEY$ nie czeka na przesłanie znaku, podobnie jak to ma miejsce w czasie współpracy z klawiaturą),
— listingi, tzn. wydruki treści programów — nadawane za pomocą instrukcji LIST, a odbierane — INPUT lub INKEY$.

Strumień, który posłuży do przesyłania informacji w sieci, powinien być otwarty w następujący sposób:

10 OPEN#*nr* ; "n"; *x*

x — nadawca podaje w instrukcji OPEN numer odbiorcy, odbiorca natomiast numer nadawcy. Numer równy zero oznacza, że transmisja jest adresowana do wszystkich komputerów włączonych w sieć, podobnie jak przy instrukcji SAVE* (patrz p. 6.1)

nr — numer strumienia danych

Instrukcje PRINT oraz LIST przesyłają dane do bufora (tworzonego w czasie otwierania strumienia) o pojemności 256 znaków. Gdy bufor jest pełen lub gdy wykona się instrukcję CLOSE, to zawartość bufora jest przesyłana przez łącze sieci. W czasie transmisji brzeg ekranu błyska. Transmisję można przerwać wciskając klawisz BREAK,N lub SPACE.

Pierwsza wykonana instrukcja: PRINT, LIST, INPUT lub INKEY$ określa, czy strumień będzie strumieniem wyjścia czy wejścia. Nie może on być jednocześnie wejściem i wyjściem dla danych.

Uwaga *Nie wolno wyłączać zasilania Spectrum, gdy trwa transmisja w sieci lokalnej, gdyż zakłóca to jej pracę.*

Największą zaletą takiej sieci jest możliwość używania urządzeń zewnętrznych, np. drukarki, microdrive lub stacji dysków dołączonych do jednego z komputerów przez wszystkie maszyny włączone do sieci.

Poniżej podano program dla komputera, który odbiera informacje od innych dołączonych do sieci i drukuje je na drukarce (odbiorca) oraz procedury dla innych nadawców, które zlecają odbiorcy wykonanie wydruków. Dla odbiorcy wybrano numer 64. Natomiast nadawcy zmieniają na chwilę swoje numery na 63 i jako komputer nr 63 przesyłają swój prawdziwy (poprzedni) numer. Dzięki temu dowolny nadawca oraz odbiorca mogą się skomunikować, mają bowiem na początku transmisji ustalone numery. Zmiana numeru nadawcy na 63 jest chwilowa i służy tylko do przesłania odbiorcy prawdziwego numeru nadawcy, potrzebnego do dalszej współpracy.

Odbiorca:

```
   5 OPEN #3;"t"
  10 FORMAT "n";64
  20 OPEN #4;"n";63
  30 INPUT #4;a: CLOSE #4
  40 LPRINT "Infomacje z kompute
ra o numerze ";a
  50 MOVE "n";a TO #3
  60 GO TO 20
```

Odbiorca stale czeka na zlecenie drukowania, otwiera strumień do czytania z komputera 63 i odczytuje numer komputera, który zleca drukowanie (linie 20, 30). Następnie powtórnie zamyka, a następnie otwiera strumień do czytania, tym razem do współpracy z komputerem o wczytanym instrukcją INPUT numerze. Dzięki temu zabiegowi nadawca zmienia swój numer na 63 na krótki okres czasu. Nie ma wtedy możliwości pomieszania danych nadchodzących z różnych maszyn. W linii 50 następuje otwarcie kanału, transmisja znaków z komputera o znanym (zapisanym w zmiennej A) numerze oraz zamknięcie kanału.

Nadawca:

```
9000 LET a=PEEK (23749): REM Zap
amietanie numeru nadawcy
9010 FORMAT "n";63: REM Ustawien
ie numeru tymczasowego=63
9020 OPEN #4;"n";64
9030 PRINT #4;a: REM Przeslanie
numeru nadawcy do odbiorcy
9040 CLOSE #4
9050 FORMAT "n";a: REM Odtworzen
ie starego numeru nadawcy
9060 OPEN #4;"n";64
9070 LIST #4
9075 REM lub PRINT #4; ...
9080 REM Przesylanie informacji
,ktore maja byc wydrukowane
9090 CLOSE #4
9100 RETURN
```

Wyróżniony komputer może także obsługiwać microdrive lub stację dysków elastycznych i realizować zlecenia innych komputerów dotyczące współpracy.

Instrukcje CLEAR, CLS 6.2.7

Instrukcja CLEAR # powoduje zamknięcie wszystkich strumieni, a następnie otwarcie strumieni: 0÷3 dla standardowych kanałów: „k", „k", „p", „s" (podobnie jak to miało miejsce po włączeniu zasilania).

Instrukcja CLEAR # wykonuje także wszystkie te czynności co CLEAR (patrz p. 2.4.3).

Uwaga *Zamknięcie strumienia związanego z microdrive'em lub siecią lokalną za pomocą CLEAR nie powoduje przesyłania informacji znajdujących się w buforze na kasetkę (do sieci), jak to ma miejsce w czasie wykonania instrukcji CLOSE #nr.*

Instrukcja CLS# jest równoważna pięciu instrukcjom:

PAPER 7 : INK 0: BRIGHT 0 : FLASH 0 : CLS

Wygodnie jest rozpoczynać każdy program linią:

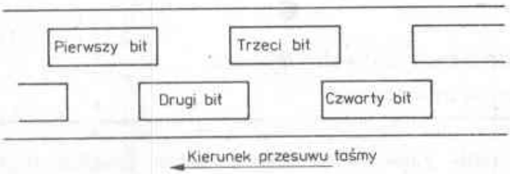
1 CLEAR# : CLOSE#

jednak można obyć się bez tych instrukcji.

* * * Budowa i zasada działania ZX Microdrive 6.3

Budowa i sposób podłączenia ZX Microdrive 6.3.1

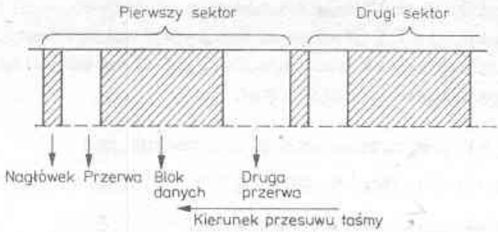
ZX Microdrive jest pamięcią zewnętrzną komputera, która pod pewnymi względami przypomina magnetofon, pod innymi zaś stację dysków elastycznych. Informacje zapisywane są na tasiemce magnetycznej zawartej w specjalnej kasetce. W kasetce znajduje się około pięciu metrów bardzo wąskiej i cienkiej taśmy sklejonej w pętlę. Informacje zapisywane są na dwu ścieżkach (rys. 30).



Rys. 30. Sposób zapisu informacji na tasiemce microdrive

Microdrive ma bardzo prostą konstrukcję. Taśma przesuwa się tylko w jedną stronę, zawsze z tą samą prędkością (w czasie zapisu, odczytu i przesuwania). Nie ma potrzeby przewijania tasiemki, gdyż stanowi ona pętlę i jest krótka. Głowice odczytujące i kasujące przypominają zwykłe magnetofonowe głowice stereo.

Dane zapisywane są na kasetce w porcjach o stałej długości — tzw. s e k t o r a c h (rys. 31). Sektor zawiera kilkanaście bajtów używanych przez system operacyjny oraz 512 bajtów danych. Między sektorami znajdują się przerwy. Informacje zawsze odczytuje się lub zapisuje całymi sektorami. Dowolny sektor może być zawsze dostępny. Na kasetce mieści się ok. 200 sektorów, tj. ok. 90 kB.



Rys. 31. Sektory zapisane na tasiemce microdrive

Pojemność używanych kasetek jest zwykle mniejsza na skutek następujących z czasem uszkodzeń tasiemki.

Microdrive podłącza się do Interface 1 przez szynę wyprowadzoną z boku tego układu. Szyna zawiera czternaście wyprowadzeń (pięciu nie używa się):

— 0 V (masa),
— 9 V (zasilanie),
— kasowanie,
— odczyt/zapis,
— 2 linie sterujące wyborem microdrive'u oraz włączaniem i wyłączaniem silnika,
— 2 linie danych (dwukierunkowe),
— zabezpieczenie kasetki przed zapisem (gdy w kasetce wyłamie się specjalny, plastykowy ząbek na linii tej pojawia się sygnał blokady zapisu).

Microdrive jest podłączony do Spectrum jako układ wejścia-wyjścia o adresach 239 (EFH), 247 (F7H) i 231 (E7H). Uaktywniany jest, gdy linie adresowe A4 lub A5 znajdują się w stanie niskim.

Struktury danych używane przez microdrive 6.3.2

W czasie otwierania pliku w obszarze rekordów kanałów (CHANS — patrz p. 4.6) rezerwuje się obszar przeznaczony na rekord kanału Microdrive, a w pole STRMS wpisuje się adres tego pola. Dla każdego otwieranego pliku tworzy się oddzielny rekord kanału*).

Rekord kanału „m" ma postać pokazaną w tabl. 5.

Adres początku rekordu pliku otwartego dla strumienia o numerze *n* można wyznaczyć za pomocą funkcji:

```
10 DEF FN A(n) = PEEK(23574+2*n)+256*PEEK(23574+
              +2*n+1)+PEEK(23631)+256*PEEK(23632)
              -1
```

*) Dzieje się więc inaczej niż w przypadku otwierania strumieni dla kanałów standardowych, gdyż otwarcie takiego strumienia nie powoduje tworzenia nowego rekordu.

Tablica 5. Postać rekordu kanału „m"

Numer bajtu	Opis
0, 1	liczba 8 — adres procedury obsługi błędu
2, 3	liczba 8 — j.w.
4	„m" — nazwa kanału
5, 6	adres procedury zapisu informacji do kanału
7, 8	adres procedury odczytu informacji z kanału
9, 10	liczba 595 — długość rekordu kanału (rekordy są zmiennej długości, gdy podłączony jest układ Interface 1)
11, 12	CHBYTE — licznik wskazujący następny bajt w buforze
13	CHREC — numer bufora (sektora) w pliku
14÷23	CHNAME — nazwa pliku (do 10 znaków)
24	CHFLAG — znacznik: $b_0 = 0$ oznacza plik otwarty do czytania, $b_1 = 1$ plik otwarty dla pisania
25	CHDRIV — numer microdrive
26, 27	CHMAP — adres mapy microdrive'u (mapa jest blokiem pamięci, w którym są zapisane informacje o tym, które sektory kasetki są zajęte, a które wolne)
28÷594	komplet informacji, które tworzą cały sektor na kasetce, w tym np. bufor danych (informacje te mają identyczną strukturę jak opisany dalej sektor); w czasie zapisu sektora na taśmie zapisuje się tę część rekordu i stanowi ona sektor (analogicznie przy odczycie sektora z kasetki)

Za pomocą tej funkcji łatwo można odczytać dowolne pole rekordu. Na przykład, za pomocą instrukcji:

```
PRINT PEEK (Fn A(n)+11)+256*(Fn A(n)+12)
```

drukuje się wartość licznika bajtów CHBYTE w pliku otwartym dla strumienia o numerze *n*.

Pierwsze 11 bajtów rekordu kanału zawiera informacje, które zawsze są umieszczane w rekordzie kanału ZX Spectrum rozszerzonego o Interface 1. Pozostałe pola są specyficzne tylko dla rekordu kanału typu „m". Informacje przesyłane za pomocą INPUT, PRINT itd. umieszczane są w buforze (512-bajtowym) znajdującym się w rekordzie kanału. Pole CHBYTE (bajt 11 i 12 rekordu) pokazuje, który bajt bufora był ostatnio zapisany lub odczytany. Jeśli w czasie operacji we/wy licznik CHBYTE ma wartość większą od 512 znaczy to, że należy zapisać (lub odczytać) następny sektor na kasetkę. Można to wykorzystać np.: do przejrzenia sektorów zajętych przez plik. Należy w tym celu wykonać np.

instrukcję INPUT — wczytanie sektora. Następnie, jeśli zmieni się licznik CHBYTE (np. na 520) i wykona powtórnie INPUT, to odbędzie się wczytanie kolejnego sektora, itd.

Gdy otwiera się plik, to system operacyjny odczytuje całą zawartość kasetki i tworzy tzw. mapę Microdrive'u*). Zajmuje ona 52 bajty i zawiera na poszczególnych bitach informacje, czy kolejne sektory są zajęte (jedynka) czy wolne (zero). Mapę Microdrive'u tworzy się przy każdym otwarciu pliku zakładając, że kasetka była wymieniana. Mapa jest potrzebna systemowi operacyjnemu, nie może on bowiem (brak czasu) odczytać, czy sektor jest wolny, a jednocześnie zapisać do niego informacje. Adres mapy pliku otwartego dla strumienia o numerze *n* można uzyskać za pomocą zdefiniowanej wyżej funkcji:

PRINT PEEK (FN A(n)+26)+256* PEEK(FN A(n)+27)

Tablica 6. Struktura sektora

Numer bajtu		Opis
Nagłówek	28÷39	12 bajtów sygnału początku sektora (10 bajtów o wartości 0, dwa o wartości 255)
	40	HDFLAG — znacznik: $b_0 = 1$ ozn. nagłówek
	41	HDNUM — numer sektora
	42, 43	nie używane
	44÷53	HDNAME — nazwa kasetki (do 10 znaków)
	54	HDCHK — suma kontrolna nagłówka (14 poprzednich bajtów)
Deskryptor	55÷66	12 bajtów sygnału początkowego (10 bajtów o wartości 0, 2 bajty równe 255)
	67	RECFLAG — znacznik: $b_0 = 0$ ozn. deskryptor
	68	RECNUM — numer bloku w pliku
	69, 70	RECLEN — długość bloku (blok zawiera maksymalnie 512 bajtów danych, lecz może zawierać mniej, np. ostatni blok w pliku)
	71÷80	RECNAME — nazwa pliku (do 10 znaków)
	81	DESCHK — suma kontrolna deskryptora
Dane	82÷593	bufor danych (512 bajtów)
	594	DCHK — suma kontrolna danych

*) Jest ona umieszczana w obszarze mapy microdrive'u, który zaczyna się od komórki o adresie 23734 a kończy w miejscu wskazanym przez zmienną systemową CHANS.

Sektor składa się z nagłówka i bloku danych. Z kolei blok danych składa się z deskryptora (opisu) oraz bufora. Nagłówek oraz deskryptor mają identyczną budowę. Strukturę sektora opisano w tabl. 6 (przy opisie podano numery bajtów, jakie mają w rekordzie kanału poszczególne pola sektora).

System operacyjny tworzy w pamięci rekord kanału nie tylko w czasie wykonywania instrukcji OPEN. Na czas wykonania instrukcji MOVE, LOAD* czy SAVE* tworzy się tymczasowy rekord kanału.

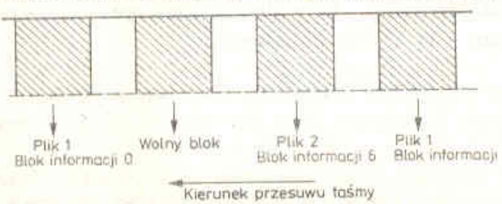
Pliki utworzone instrukcją SAVE* nie mogą być odczytane za pomocą INPUT czy INKEY$, lecz jedynie za pomocą LOAD* lub MERGE*. Jeśli jednak programista chce odczytać taki plik za pomocą np. INKEY$, to może „oszukać system operacyjny". Plik z programem należy otworzyć za pomocą instrukcji OPEN, a następnie w pole RECFLAG (64 bajt rekordu kanału) wpisać liczbę 4.

Pliki utworzone instrukcją SAVE (zwane z ang. *non-PRINT*) w pierwszym rekordzie, w 512-bajtowym bloku danych zawierają nagłówek podobny do zapisywanego na kasecie magnetofonowej (tabl. 7).

Tablica 7. *Nagłówek programu zapisanego na kasetce microdrive'u*

Numer bajtu w bloku	Opis
1	znacznik: 0 — program w Basicu, 1, 2 — tablica w Basicu, 3 — ciąg bajtów (kod binarny lub pamięć ekranu)
2, 3	długość pliku
4, 5	adres początku pliku w pamięci
6, 7	długość obszaru programu, dla programów napisanych w Basicu
8, 9	adres startu (nr linii) dla samostartującego programu w Basicu

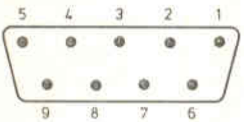
Plik, który zawiera wiele 512-bajtowych bloków nie jest zapisywany w kolejnych sektorach znajdujących się na tasiemce microdrive'u. Sektory pliku rozrzucone są na całej jej długości. Organizacja kasetki wyglądać może tak jak na rys. 32.



Rys. 32. Organizacja danych na kasetce

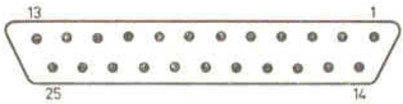
Standard RS 232 dokładnie określa zasady instalacji i działanie łącza szeregowego. Przez takie łącze (ang. *serial*) dane są przesyłane bit po bicie. Do transmisji używa się dwu linii danych — transmisja jest dwukierunkowa*⁾.

Jednak poszczególne firmy nie zawsze respektują wszystkie ustalenia standardu RS 232. Często uwzględnia się tylko jego część. Dlatego przed podłączeniem Spectrum (z dołączonym układem Interface 1) z urządzeniem wyposażonym w układ sprzęgający RS 232 należy sprawdzić, czy wartości napięć i sygnałów łącza dobrane są poprawnie.



2 - TX - wg danych
3 - RX - wg danych
4 - DTR - sygnał gotowości z urządz. zewnętrznego (stan wysoki)
5 - CTS - sygnał gotowości Spectrum (stan wysoki)
7 - Masa (0V)
9 - Zasilanie (9V)
Linie nr 1, 6, 8 - nie podłączone

Rys. 33.
9-stykowe gniazdo RS 232



2 - TX - wejście danych
3 - RX - wyjście danych
5 - CTS - sygnał gotowości Spectrum (stan wysoki)
6 - +9V
7 - 0V
20 - DTR - sygnał gotowości (stan wysoki) i urządzenie zewn.
Pozostałe linie nie podłączone

Rys. 34.
25-stykowe gniazdo RS 232

W Interface 1 znajduje się 9-stykowe gniazdo przeznaczone do współpracy z RS 232 (patrz rys. 33). Zastosowano tu gniazdo zamiast wtyku, aby uniemożliwić podłączenie standardowego manipulatora (joysticka) do gniazda RS 232.

Uwaga *W innych komputerach i urządzeniach zewnętrznych stosuje się niekiedy inne, np.: 25-stykowe gniazda (rys. 34) lub wtyki.*

Najczęściej kłopoty sprawia poziom napięć wymaganych na poszczególnych liniach łącza. W Spectrum przyjęto, że zero logiczne reprezentowane jest na wyjściach RS 232 napięciem −14 V (bez podłączonego obciążenia), a jedynka logiczna napięciem +14 V (bez obciążenia).

Uwaga *Jeśli urządzenia zewnętrzne mają inne napięcia niż Spectrum, to należy je dołączyć przez specjalny układ dopasowujący poziomy.*

*⁾ Alternatywą jest przesłanie całej „paczki" bitów (np. bajtu) po kilku (ośmiu) liniach danych. Takie łącze nazywa się równoległym (ang. *parallel*).

Zazwyczaj jednak urządzenia zewnętrzne, np. drukarki mają dużą tolerancję poziomów napięć na liniach łącza RS 232 i mogą współpracować ze Spectrum. Na linię 9 łącza ZX Spectrum podaje napięcie +9 V. Należy sprawdzić, czy linia ta w dołączonym do komputera urządzeniu nie jest zwarta do masy, gdyż połączenie grozi wtedy spaleniem układu Interface 1.

Informacje mogą być przesyłane przez łącze RS 232, gdy zarówno urządzenie, jak i ZX Spectrum jest gotowe do współpracy. Gotowość tę sygnalizuje podając wysokie napięcie na odpowiednią linię RS 232. Jest ona sprawdzana po przesłaniu każdego bajtu. Kolejne bajty przesyłane są bit po bicie wg następującego formatu:
— osiem bitów danych,
— dwa bity stopu.

Uwaga *Nie przesyła się bitu parzystości.*
Niektóre urządzenia mogą używać innego formatu danych, np. jeden bit stopu. Urządzenia takiego nie można dołączyć za pomocą łącza RS 232 zawartego w Interface 1.

W ZX Spectrum szybkość transmisji danych przez łącze RS 232 jest programowana i należy ją dobrać do konkretnego urządzenia. Gdy przesyłanie informacji odbywa się pod kontrolą programu napisanego w języku Basic, to szybkość przesyłania z reguły nie może przekroczyć 110÷300 bitów/s (najlepiej wybrać 50 bitów/s).

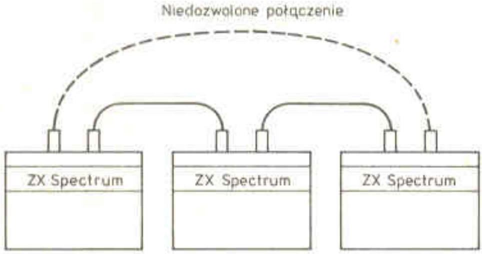
Przez łącze RS 232 ZX Spectrum można połączyć nie tylko z urządzeniem zewnętrznym, ale także z innym komputerem (w ten sposób można połączyć ZX Spectrum i Sinclair QL).

Strukturę rekordu kanałów „b" i „t" (są one identyczne) podano w tabl. 8.

***Tablica 8.** Struktura rekordu kanałów „b" i „t"*

Numer bajtu	Opis
0, 1	liczba 8 — adres procedury obsługi błędu
2, 3	liczba 8 — j.w.
4	„t" lub „b" — nazwa kanału
5, 6	adres procedury transmisji danych z komputera: 0C3CH dla „t", 0C5AH dla „b"
7, 8	adres procedury transmisji danych do komputera: 0B6FH dla „t", 0B75H dla „b"
9, 10	liczba 11 długość rekordu

W sieć można połączyć od 2 do 64 komputerów. Sieć nie może stanowić zamkniętej pętli — rys. 35. Szybkość transmisji informacji w sieci wynosi ponad 30 000 bitów/s*).



Rys. 35. ZX Spectrum połączone w sieć lokalną

Komputery są połączone dwiema liniami — linią danych oraz linią masy. W danej chwili mogą komunikować się dwa spośród komputerów dołączonych do sieci lub wszystkie mogą odbierać informacje nadawane przez jeden z nich. Dlatego też musi istnieć mechanizm wyboru komputera, który w określonej chwili może nadawać.

Mechanizm ten jest w ZX Spectrum niezwykle prosty. Wykorzystuje on tę własność łącza sieci, że jeśli choć jeden z komputerów wymusza na linii danych sieci logiczną jedynkę (napięcie 5 V), to linia ta będzie w stanie wysokim niezależnie od tego, jakie napięcie wymuszają inne komputery. Komputer, który zamierza przesłać dane sprawdza najpierw, czy w sieci przesyłane są informacje. Jeżeli sieć jest wolna rozpoczyna nadawanie. W czasie nadawania odczytuje stan linii danych. Gdy dwa komputery rozpoczną nadawanie jednocześnie, to jeden z nich stwierdzi w pewnej chwili, że wymuszał na linii stan niski, a linia pozostała w stanie wysokim. Wyłącza się wtedy i oczekuje na zwolnienie sieci. Zawsze wyłączy się komputer, który ma niższy numer, na początku transmisji bowiem nadawca przesyła swój numer.

Transmisja rozpoczyna się od nagłówka. Dane nagłówka są pobierane z rekordu kanału. Rekord taki tworzy się w czasie otwarcia strumienia dla kanału *n*. Następnie nadawca oczekuje na potwierdzenie od adresata. Jeśli potwierdzenia nie ma, to nadawca powtarza wszystkie czynności od początku, łącznie z przydziałem linii sieci. Gdy odbiorca prześle potwierdzenie (pojedynczy bajt), to rozpoczyna się transmisja danych.

*) Sieć jest „szybsza" od łącza RS 232 oraz od wielu stacji dysków elastycznych.

W sieci zawsze przesyła się pełen bufor znaków. W czasie pisania informacje są umieszczane w buforze w rekordzie kanału (tabl. 9). Gdy bufor jest pełen, to nadawca oczekuje na gotowość odbiorcy do transmisji, po czym przesyła pełen bufor. Odbiorca pobiera kolejne bajty swego bufora, a gdy go opróżni oczekuje na nowe dane z sieci.

Tablica 9. *Struktura rekordu kanału* ,,n"

Numer bajtu	Opis
0, 1	liczba 8 — adres procedury obsługi błędu
1, 3	liczba 8 — j.w.
4	,,n" — nazwa kanału
5, 6	adres procedury pisania ,,do sieci"
7, 8	adres procedury czytania ,,z sieci"
9, 10	liczba 276 — długość rekordu
11	NCIRIS — numer komputera przeznaczenia
12	NCSELF — numer komputera (w czasie wykonywania instrukcji OPEN) ustalony instrukcją FORMAT
13, 14	NCNUM — numer bloku danych
15	NCTYPE — typ danych (0 — dane, 1 — rekord końca)
16	NCOBL — liczba danych (gdy sieć otwarta do czytania, zero — gdy otwarta do pisania)
17	NCDCS — suma kontrolna nagłówka
19	NCCUR numer znaku ostatnio pobranego z bufora
20	NCIBL liczba znaków w buforze (gdy kanał otwarty dla czytania, zero gdy dla pisania)
21÷276	bufor danych (256 bajtów)

* * * Sposób działania układu Interface 1 6.6

Układ Interface 1 zawiera 8 kB pamięci stałej. W pamięci tej zapisano dodatkowe procedury systemu operacyjnego oraz programy rozszerzające możliwości interpretera języka Basic. Pamięć ta jest podłączona w miejsce pamięci ROM ZX Spectrum w obszarze przestrzeni adresowej od 0 do 8192 (0÷1FFF H). Przełączenia pamięci dokonuje układ elektroniczny zawarty w Interface 1. Przełączenie pamięci następuje w czasie pobierania rozkazu z komórek o adresie 8 oraz 5806

(1708 H)*). Podanie napięcia +5 V na linię „ROM disable" wyprowadzoną na szynie krawędziowej powoduje odłączenie pamięci stałej. W jej miejsce można podłączyć inną pamięć. Układ Interface 1 „podłącza" wtedy swój ROM.

Gdy w programie napisanym w języku Basic wystąpi instrukcja, której interpreter nie potrafi zinterpretować (jedna z instrukcji dodatkowych opisanych w rozdz. 6.2 lub poprostu pomyłka), to jest wywoływana procedura obsługi błędu. Wywołanie to ma w języku asemblera postać:

RST 8	wywołanie procedury o adresie 8
DEFB *nr*	numer błędu

Wykonanie RST 8 umożliwia zamianę pamięci stałej ZX Spectrum na ROM z Interface 1. Programy zapisane w nowej pamięci analizują instrukcję i jeśli jest ona rozszerzeniem Basica interpretują ją. Jeżeli instrukcja jest niepoprawna, to wykonywany jest skok w miejsce pamięci, którego adres jest pobierany ze zmiennej systemowej VECTOR (2 bajty o adresie 23735,23736)**). Zmienna VECTOR zawiera zazwyczaj adres procedury obsługi błędu — 271 (01F0 H). Można jednak wpisać tam inną wartość. Pozwala to na rozszerzenie języka Basic. Należy napisać procedury interpretujące nowe instrukcje a ich adres zapamiętać w zmiennej VECTOR.

Liczby podawane jako następny bajt po instrukcji RST 8 oznaczają zazwyczaj kod błędu. Liczby o wartościach od 27 (1B H) do 50 (32 H) mają dla ROM-u z Interface 1 specjalne znaczenie. Informują one, jaka procedura w nowym ROM-ie ma być wywołana. Noszą one nazwę z ang. *hook code*.

Wywołanie procedury zawartej w ROM-ie Interface 1 ma postać w języku asemblera:

```
RST 8 hook code
```

Przykładowe kody tego typu podano w tabl. 10.

Hook code równy 50 (32 H) pozwala na wywołanie dowolnej procedury zawartej w pamięci stałej Interface 1. Adres procedury pobiera się ze zmiennej systemowej HD-11 (bajty o adresie 23789,23790). Dokładniejsze informacje o hook code'ach znaleźć można w pracy [3].

Programów zawartych w dodatkowym ROM-ie nie można odczytać za pomocą funkcji PEEK lub programu disasemblera. Jedynym sposobem jest przepisanie pamięci stałej z Interface 1 na kasetkę Microdrive, a następnie do pamięci RAM:

```
10 CLEAR 35000
20 SAVE* "m"; 1; "ROM" CODE 0,8192
30 LOAD* "m"; 1; "ROM" CODE 35000,8192
```

*) Ta druga możliwość pozwala poprawić błąd w interpreterze Basica. Otóż w ZX Spectrum wykonanie instrukcji CLOSE dla strumienia o numerze większym od 3 może doprowadzić do wyzerowania systemu.

**) Po dołączeniu Interface 1 rozszerzony system używa dodatkowych zmiennych systemowych, które zajmują 58 bajtów pamięci o adresach 23734 do 23791.

Tablica 10. Wybrane hook code'y

Hook code	Opis
33	włącz silnik microdrive'u, akumulator A w procesorze zawiera numer microdrive'u; 0 oznacza wyłącz wszystkie silniki
34	otwórz tymczasowy kanał
37	odczytaj następny blok danych z pliku
38	zapisz następny blok danych do pliku
41	odczytaj następny sektor taśmy microdrive'u (jest on ładowany do bufora zawartego w rekordzie kanału)
27	wczytaj znak wciśnięty na klawiaturze do akumulatora
28	wprowadź znak znajdujący się w akumulatorze do strumienia nr 2
29	wczytaj bajt przesłany przez RS 232 do akumulatora; jeśli bajt jest dostępny, to C = 0

Program umieszczony w pamięci RAM można „obejrzeć" posługując się disasemblerem (patrz 8.10). Program umieszczony w Interface 1 został zastrzeżony i jego kopiowanie jest bezprawne.

Pamięci dyskowe do ZX Spectrum

* 7

Pamięć masowa do ZX Spectrum stosowana przez firmę Sinclair czyli ZX Microdrive jest, ze względu na małą trwałość oraz szybkość działania, krytykowana przez użytkowników. W zamierzeniach miała być, dzięki niskiej cenie, konkurencyjna wobec pamięci dyskowej, jednak spadek cen dysków zniweczył te zamiary. Ponieważ Sinclair uparcie stosuje Microdrive, pamięci dyskowe do ZX Spectrum (a także Sinclaira QL) zaczęły produkować inne firmy. Pamięci te funkcjonalnie (pod względem możliwości i instrukcji) przypominają Microdrive. Podobny jest format instrukcji i sposób współpracy z ZX Spectrum. Sinclair, projektując Microdrive, wzorował się na dyskach, z kolei autorzy dysków do Spectrum naśladowali Microdrive. Dlatego dyskom poświęcono stosunkowo mało miejsca, mimo że wypierają z użycia Microdrive.

Dysk elastyczny (ang. *floppy disc, diskette*) to krążek elastycznego plastiku pokryty warstwą tlenku żelaza i umieszczony w ochronnej kopercie. Sposób zapisu na dyskietce jest taki sam jak na taśmie magnetofonowej czyli magnetyczny. Na świecie rozpowszechnione są dyski elastyczne o średnicy 8, 5 1/4, 3 1/2 i 3 cale. Dzięki szybkiemu dostępowi do danych, dużej pojemności i niezawodności są one podstawowym rodzajem pamięci zewnętrznej dla wszystkich komputerów osobistych.

Zapisu i odczytu danych na dysku dokonuje się za pomocą stacji dysków (ang. *disk drive* — czyli napęd). Dyskietka umieszczona w stacji wiruje ze stałą prędkością, do odczytu i zapisu służy głowica działająca tak jak magnetofonowa. Przekazywaniem danych z i do komputera steruje układ tzw. kontrolera. Dane zapisywane są na dysku elastycznym blokami o stałej długości — sektorami. Kolejne sektory zapisywane są na koncentrycznych kręgach tzw. ścieżkach. Podobnie jak w microdrive transmituje się całe sektory (co najmniej). Głowica porusza się wzdłuż promienia dyskietki, dzięki czemu może być przesunięta na dowolną ścieżkę, co pozwala szybko odszukać żądaną porcję informacji. Na jednym dysku elastycznym znajduje się 40 lub 80 ścieżek (zależnie od konstrukcji napędu), z których każda zawiera od 8 do 16 sektorów. Zazwyczaj dane (aby pomieścić ich jak najwięcej) są kodowane w systemie o nazwie MFM. Jeden sektor

mieści w takim przypadku od 128 do 1024 bajtów. Na dyskietce można zatem zapisać (w zależności od stosowanej organizacji i napędu) od 100 do 360 kB. Istnieją systemy, które zapisują dane na obu stronach dysku.

Oprogramowanie obsługujące pracę stacji dysków tzw. system dyskowy (DOS z ang. *disc operating system*) realizuje funkcje: zapisu i odczytu danych, programów, sektorów, manipulowania zbiorami informacji: kopiowania, zmiany nazwy, usuwania, ochrony informacji przed dostępem przez nieuprawnionego użytkownika itd.

W odróżnieniu od microdrive'ów informacja o wszystkich zapisanych zbiorach przechowywana jest stale na dyskietce, a więc aby odszukać żądany plik nie trzeba przeglądać całej zawartości dysku.

Pamięci dyskowe do ZX Spectrum produkowane są przez wiele firm. Poszczególne modele znacznie różnią się między sobą, bowiem firmy tworzą zwykle własne systemy współpracy z tym rodzajem pamięci. Dotyczy to zarówno sprzętu (układy sprzęgające, kontrolery, napędy), jak i oprogramowania (systemy dyskowe). Układy te oferują prócz współpracy z dyskietką także dodatkowe możliwości np. łącze szeregowe RS 232 lub równoległe Centronics, sieć lokalna bądź współpracę z manipulatorem (ang. *joystick*) — podobnie jak Interface 1. Do Spectrum dołączane są zwykle stacje dysków 5 1/4", używane są także dyski 3" i 3 1/2". W systemach rozpowszechnionych w Polsce na jednej stronie dyskietki zapamiętuje się zwykle ok. 150 kB.

Współpraca między systemem operacyjnym ZX Spectrum a pamięcią dyskową odbywa się tak, jak przy podłączeniu Interface 1, przez podmianę części pamięci ROM (patrz 6.6). Niektóre systemy mają także własną pamięć RAM (na bufor danych), a nawet własny procesor. Zazwyczaj jeden układ kontrolera może obsługiwać kilka stacji dyskowych. Pamięci dyskowe do ZX Spectrum zawsze wyposażone są w odrębny zasilacz.

W Polsce stacje dysków do Spectrum oferuje wiele firm. Sprzedają je firmy polonijne, np. Polbrit (dyski 5 1/4" i 3") i Apina. Oferują je także firmy państwowe i rzemieślnicze. Łączy je przede wszystkim jedno — wysoka cena (ok. dwukrotnie wyższa niż samego komputera). Oprogramowanie (dyskowe systemy operacyjne) z reguły powstało za granicą — w Wielkiej Brytanii lub USA.

W dalszej części rozdziału przedstawione zostaną dwie wybrane realizacje — dyski 5 1/4" i 3" rozprowadzane przez Polbrit.

Pamięć dyskowa 5 1/4 cala

W skład zestawu wchodzi interfejs wraz z kontrolerem i stacja dysków z wbudowanym zasilaczem. Oprogramowanie zapewniające współpracę ZX Spectrum z dyskiem jest oparte na brytyjskim systemie TR-DOS. W układzie sterowania zastosowano układ scalony WD 1793, co umożliwiło stosowanie standardowego zapisu MFM (tzw. podwójna gęstość zapisu — ang. *double density*). Przy podziale

powierzchni dysku na 40 ścieżek po 16 sektorów o długości 256 bajtów jego pojemność wynosi 160 kB na jednej stronie dyskietki. Jedna ze ścieżek jest wykorzystywana przez system na katalog, informacje o położeniu zbiorów itd. W tym systemie kontroler steruje pracą tylko jednego napędu.

Zastosowano węgierską stację dysków elastycznych M1800/900, która umożliwia dostęp tylko do 35 ścieżek, co zmniejsza pojemność dysku do ok. 140 kB. System dyskowy TR-DOS jest zapisany w dodatkowej pamięci stałej zawartej

Tablica 11. *Zestawienie instrukcji obsługi pamięci masowych dla ZX Spectrum*
[ ] — *parametry opcjonalne*

Operacje	Microdrive
Formatowanie	FORMAT „m"; *nr*; „*nazwa*"
Zapisywanie programu	SAVE* „*nazwa*" [CODE, DATA SCREEN$, LINE]
Wczytywanie programu	LOAD* „*nazwa*" [DATA, CODE, SCREEN$] MERGE* „*nazwa*"
Otwarcie pliku	OPEN# *nr*; „m"; *nr*; „*nazwa*"
Zamknięcie pliku	CLOSE# *nr*
Zapisywanie danych	INPUT# *nr*, PRINT# *nr*, LIST#
Wczytywanie danych	INPUT# *nr*, INKEY$# *nr*
Katalog	CAT
Usunięcie pliku	ERASE „*m*"; nr; „*nazwa*"
Kopiowanie pliku	MOVE *źródło* TO *cel*
Ustawienie atrybutów pliku	—
Zmiana bieżącego katalogu	—
Utworzenie katalogu lub pliku	—
Przewinięcie pliku	—
Zwolnienie miejsca na nośniku	automatyczne po usunięciu pliku
Zmiana hasła	nie potrzeba
Powrót do Basicu	nie potrzebny

w układzie sprzęgającym. Podmienia ona 8 kB pamięci ROM ZX Spectrum. TR-DOS wykorzystuje także 112 bajtów pamięci RAM Spectrum rozszerzając pole zmiennych systemowych. Po włączeniu zasilania system natychmiast zgłasza się wyświetlając nagłówek i pytając o hasło (odbywa się to bez uruchomienia stacji dysków). Niezależnie od odpowiedzi system przechodzi do fazy oczekiwania na instrukcje. Jeśli jednak hasło było błędne nie są one wykonywane. Pełny zestaw instrukcji podano w tablicy 11.

Dysk 5¼"	Dysk 3"
—	FORMAT ,,*nazwa napędu*'' TO ,,*nazwa dyskietki*''
SAVE* ,,*nazwa*''[CODE, DATA LINE, SCREEN$]	SAVE* ,,*nazwa*'' CODE, DATA, SCREEN$, LINE] [N]
LOAD* ,,*nazwa*'' [CODE, DATA]	LOAD* ,,*nazwa*'' [CODE, DATA, SCREEN$] MERGE* ,,*nazwa*''
—	OPEN #*nr; ,,*nazwa*''; tryb [;długość]
—	CLOSE # *nr*
—	PRINT *# *nr* [AT *nr*] (tylko tekst)
	INPUT * # *nr* (tylko tekst)
CAT	CAT * CAT * ,,*nazwa katalogu*'' LIST *
ERASE ,,nazwa'' CODE, DATA	ERASE * ,,*nazwa*'' [N]
—	MOVE * ,,*źródło*'' TO ,,*cel*''
—	ATTR * ,,*nazwa*'' [P,U,V,I]
—	GO TO * ,,*nazwa*'' GO SUB * ,,*nazwa*'' DRAW *
—	DIM * ,,*nazwa*''
—	RESTORE * # *nr*
MOVE	automatyczne po usunięciu pliku
USR	nie potrzebna
RETURN	nie potrzebny

System TR-DOS należy uznać za wyjątkowo prymitywny, choć zajmuje 8 kB. Lista rozkazów jest bardzo uboga. Nie zawiera np. instrukcji formatowania (wstępnego podziału powierzchni dysku na ścieżki i sektory). Nie można zatem użyć dowolnych dyskietek, lecz tylko zaformatowanych, sprzedawanych przez firmę rozprowadzającą ten system za wyższą, niż normalna, cenę. (Za dodatkową opłatą Polbrit sprzedaje specjalny program formatujący). Ochrona zbiorów przed dostępem przez niepowołane osoby zapewniana jest tylko przez podanie wstępnego hasła. Usunięcie zbioru z dysku nie oznacza zazwyczaj zwiększenia liczby wolnych sektorów. Konieczne jest jeszcze wykonanie instrukcji MOVE, która porządkuje zawartość dysku.

Niewygodna jest także współpraca z programem napisanym w Basicu. Linia wykonująca instrukcje systemu TR-DOS musi mieć postać:

RANDOMIZE USR 15363:REM: *instrukcja*

Przy wywołaniu systemu dyskowego z Basica nie są sygnalizowane błędy wykonania (np. z powodu uszkodzenia nośnika). Dane, np. liczby lub znaki nie mogą być zapisywane i odczytywane z dysku za pomocą instrukcji PRINT i INPUT. Można zapamiętywać jedynie programy — LOAD i SAVE. Program zapisany na dysku może być uruchomiony natychmiast po załadowaniu do pamięci, jeżeli zostanie wykonana instrukcja RUN "*nazwa*".

Transmisje dyskowe wykonują się dość szybko. Na przykład zapamiętanie bloku 30.000 bajtów zajmuje ok. 10 s, jego odczytanie ok. 7 s.

Pamięć dyskowa 3-calowa

Ten bardzo interesujący system jest dziełem firmy Timex rozprowadzającej komputery Sinclaira w USA. Umożliwia on podłączenie do Spectrum równocześnie do 4 stacji dysków sterowanych jednym układem kontrolera. Pozwala to na zarządzanie zbiorem danych zajmujących około 500 kB (pojedyncza dyskieta mieści 160 kB). Korzystanie z tak dużej ilości danych nie jest trudne dzięki bardzo dobremu dyskowemu systemowi operacyjnemu o nazwie TOS. System ten umożliwia dodatkowo współpracę z urządzeniami zewnętrznymi za pośrednictwem dwu łącz szeregowych RS 232. Opisywane dyski mogą być zastosowane do komputerów innego typu, np. Timex 2048 lub Commodore 64 (po zastosowaniu innego interfejsu).

Całe urządzenie składa się z dołączanego do ZX Spectrum (do szyny krawędziowej) niewielkiego układu sprzęgającego, kontrolera, zasilacza i samych napędów (jeden zasilacz może obsłużyć tylko dwa napędy). Sam napęd jest identyczny ze stosowanym w komputerze domowym AMSTRAD 664. Układ kontrolera śmiało można uznać za oddzielny komputer. Ma własny procesor (Z80), 16 kB pamięci, sterownik dysku (WD1770) i inne układy wejścia-wyjścia. Dzięki temu nie zajmuje pamięci ZX Spectrum i może realizować bardzo bogaty zestaw instrukcji dotyczących zarówno dysków jak i urządzeń podłączonych za pośrednictwem łącza RS 232.

Dyskietka 3-calowa chroniona jest sztywną plastikową kopertą. Wszystkie otwory są zakryte, warstwa magnetyczna jest bowiem wielokrotnie cieńsza od ziarnka kurzu, a więc niezwykle łatwo ją uszkodzić. Napęd ze względu na konieczną precyzję ruchu głowicy (bardzo wąskie ścieżki) jest bardzo czuły na gwałtowne przemieszczenia, ruchy itp. Zapisywana jest jedna powierzchnia dyskietki (po przełożeniu dyskietki na drugą stronę można z niej dalej korzystać). W opisywanym systemie jest ona podzielona na 40 ścieżek, wielkość sektora wynosi 256 bajtów, więc na jednej stronie dyskietki mieści się 160 kB, z czego system zajmuje ok. 20 kB. Stosuje się zapis MFM.

System TOS realizuje wiele instrukcji, które tak jak w Interface I są rozszerzeniami instrukcji Basica ZX Spectrum. Wykaz instrukcji dotyczących dysku zamieszczono w tablicy 11. Instrukcje te pozwalają na zapisywanie programów i danych oraz na korzystanie z mechanizmu przesyłania danych (znakowych) do zbiorów dyskowych za pośrednictwem strumieni i kanałów. (Jest on zbliżony do stosowanego w Interface 1 i ZX Microdrive. Kanały nie pokrywają się ze stosowanymi w ZX Spectrum, nie są też wykorzystywane standardowe strumienie). Zbiory utworzone na dysku, bądź w pamięci ZX Spectrum mogą być transmitowane do innych komputerów lub urządzeń poprzez łącze szeregowe.

Zaletą tego systemu dyskowego jest hierarchiczny system zbiorów. Katalog, czyli spis zbiorów zapisanych na dysku może zawierać podkatalogi, te z kolei następne podkatalogi itd. (np. w katalogu *Książki* w podkatalogu *Komputery* znajdzie się zbiór *Przewodnik po ZX Spectrum*). Taka struktura zbiorów (dzięki bogatemu zestawowi instrukcji) ułatwia korzystanie z informacji zapisanych na dyskietce*). Kolejne instrukcje pozwalają na określanie atrybutów zbiorów (zezwolenie na zapis i odczyt, zezwolenie na wyświetlanie nazwy zbioru instrukcją CAT *), formatowanie dysku, określanie parametrów transmisji szeregowej, kopiowanie zbiorów lub grup zbiorów, wyświetlanie informacji o zbiorach i kanałach. Postać rozszerzonych instrukcji przypomina format stosowany w Interface 1 — po słowie kluczowym należy dopisać znak „*". System nie rozróżnia nazw pisanych małymi i dużymi literami.

System taki jak ten określa się ang. terminem „user friendly" — łatwy w obsłudze, np. jeden ze zbiorów na firmowym dysku zawiera informacje o wszystkich instrukcjach TOS'a. Możliwe jest pomijanie części nazwy zbioru przy korzystaniu z dysku. Jest ona zastępowana jednym znakiem: „+" lub „?", np. jeśli napisze się "?.bas" uzyska się dostęp do wszystkich zbiorów zapisanych jako basicowe. Można także zapisać na dysku program samostartujący, który będzie zawsze

) W związku z hierarchiczną organizacją zbiorów na dysku należy podać (np. w instrukcji LOAD) oprócz nazwy pliku nazwę katalogu, w którym się on znajduje oraz nazwy podkatalogów. Np. załadowanie programu z dysku demonstracyjnego : LOAD*"SUN:GAMES:HI-LO.BAS" oznacza wywołanie z katalogu SUN i podkatalogu GAMES programu HI-LO.BAS. Słowo BAS umieszczone po kropce jest opcjonalnym rozszerzeniem nazwy i oznacza, że plik jest programem w języku Basic.

wykonywany w momencie startu systemu po włączeniu zasilania lub naciśnięciu klawisza RESET (program o nazwie „start", podobnie jak „run" w ZX Microdrive).

Opisywane dyski pracują dość szybko, np. przesłanie bloku o długości 30000 bajtów zajmuje ok. 15 s, formatowanie dyskietki ok. 30 s, a 10-krotne przesłanie 220 znaków do zbioru bezpośredniego dostępu (zbiór do którego można zapisywać i odczytywać dane) zajmuje ok. 1 min.

System dyskowy sprzedawany jest łącznie z dyskiem demonstracyjnym, na którym oprócz systemu TOS znajduje się szereg bardzo przydatnych programów użytkowych i przykładowych. Dołączana jest także szczegółowa instrukcja, w której znaleźć można nawet opis ważniejszych procedur zawartych w dodatkowym ROM'ie. Ogólnie system ten uznać można za dobry, znacznie lepszy od dysków pięciocalowych.

** Nie tylko Sinclair Basic 8

* Wprowadzenie 8.1

Nie bez powodu przytłaczająca większość mikrokomputerów produkowanych obecnie na świecie jest sprzedawana łącznie z interpreterem języka Basic, umieszczonym na ogół w wydzielonym miejscu pamięci stałej (ROM). Język ten, mimo swoich wad, ma wielu zwolenników wśród programistów. Umożliwia bowiem łatwe testowanie i uruchamianie programów. Jest interpretowany, więc wyniki działania programu otrzymuje się natychmiast po jego zapisaniu. Ma liczne instrukcje dotyczące przetwarzania tekstów, obrazów i dźwięków. Poza tym jego translator (program tłumaczący instrukcje Basica na kod wewnętrzny) jest stosunkowo łatwy do napisania dla danej konfiguracji sprzętu (w informatyce mówi się: łatwy do implementacji). O ile interpreter jest zapisany w pamięci ROM, to nie trzeba go, przy błędnym wykonaniu programu, ładować powtórnie do pamięci. Sinclair Basic do tych zalet dodaje jeszcze jedną — precyzyjny sposób eliminowania błędów składni instrukcji przy wprowadzaniu programu do pamięci.

Czy wobec tego można uznać, że zapewnia on wszystko, co jest potrzebne do efektywnego programowania komputera?

Taki wniosek byłby co najmniej pochopny. Ma on wiele wad, przede wszystkim: powolność wykonania programu i nieelegancki, z punktu widzenia informatyki, niestrukturalny zapis. Twierdzi się więc, że uczy on złych nawyków w programowaniu. Zdaniem J. Weizenbauma, profesora informatyki w sławnym Massachusetts Institute of Technology (Byte 9/84): „Basic z pedagogicznego punktu widzenia jest intelektualnym monstrum", nie nadającym się kompletnie do nauki programowania.

Dla tych, którzy poznali tylko ten język, poglądy autorów mogą wydać się gołosłowne. Pozostaje nadzieja, że zamieszczone w tym rozdziale informacje na temat innych, dostępnych na ZX Spectrum języków pozwolą uznać te twierdzenia za uzasadnione.

Głównym przeznaczeniem ZX Spectrum jest zabawa. Ale nie tylko. Ogromna liczba sprzedanych egzemplarzy tego komputera zachęciła firmy produkujące

oprogramowanie do napisania i rozpowszechniania realizacji*) wielu języków programowania. Implementacje te są różne, lepsze i gorsze, ale są, i to w cenach dostępnych dla przeciętnego użytkownika (15÷25$, w firmach polonijnych i przedsiębiorstwach: kilka ÷ kilkanaście tysięcy złotych, w obiegu prywatnym — kilkaset ÷ tysiąc złotych). Wszystkie mają jedną cechę wspólną — są ładowane z taśmy magnetofonowej do pamięci RAM. Ma to tę wadę, że błędne wykonanie przetłumaczonego programu niejednokrotnie wymaga ponownego wczytania programu tłumaczącego tzw. translatora.

Poniższy przegląd języków programowania wiedzie od programów interpretujących, rozszerzeń możliwości języka Basic, przez najpopularniejszy w edukacji i coraz częściej uważany za następcę Basica język Logo, do języka Forth. Następnie zostaną przedstawione języki kompilowane. Ich reprezentację stanowić będą kompilatory Basica, Pascala i C. Z kolei przedstawiony zostanie jeszcze jeden interpreter — tym razem języka bardzo wysokiego poziomu, działającego w sposób całkowicie odmienny — języka Prolog. Na zakończenie opisano wybrany Asembler dla ZX Spectrum oraz programów wspomagających uruchamianie programów asemblerowych.

Rozdział ten przeznaczony jest w zasadzie dla osób mających już pewne doświadczenie w programowaniu. Ma on za zadanie zaznajomić nie tyle z samymi językami (stąd częste odwołania do literatury), ile z ich ogólną charakterystyką i specyficznymi cechami ich implementacji na ZX Spectrum.

Ze względu na rosnące zainteresowanie szerzej przedstawiono interpreter Logo. Bardziej szczegółowo potraktowano też popularne na naszym rynku programy: Asembler, Beta-Basic i Pascal.

* Beta-Basic 8.2

Dla wielu użytkowników Basic ZX Spectrum zapisany w pamięci stałej jest niewystarczający. Rzeczywiście, w porównaniu z realizacjami tego języka na innych komputerach osobistych, jego lista instrukcji jest stosunkowo uboga. Stworzono zatem wiele programów rozszerzających interpreter Basica. Jednym z nich jest Beta-Basic. Do tej pory są rozpowszechnione dwie wersje tego programu: Beta-Basic 1.0 oraz Beta-Basic 1.8. Prócz tego jest wiele programów o innej nazwie, ale o tej samej zawartości. Wersja pierwsza wprowadza 27 nowych instrukcji oraz 10 nowych funkcji, przy tym zajmuje 5,3 kB pamięci operacyjnej. Natomiast Beta-Basic 1.8 wprowadza 30 nowych instrukcji oraz 21 nowych funkcji, zajmując około 9,3 kB pamięci operacyjnej. Obie wersje są umieszczone w górnej części pamięci RAM. Jako, że pierwsza wersja tego programu jest w całości zawarta w drugiej, dlatego zostaną pokrótce omówione wybrane instrukcje programu Beta-Basic 1.8.

*) Taka relizacja języka na konkretnym komputerze nosi nazwę implementacji.

Program Beta-Basic ładuje się do pamięci komputera instrukcją LOAD "". Po jego wczytaniu na dole ekranu wyświetla się komunikat „Beta Basic 1.8. Betasoft 1984". Znaczy to, że można przystąpić do pracy z rozszerzonym interpreterem Basica. Bezpośrednim znakiem, świadczącym o tym że wczytany program znajduje się w pamięci mikrokomputera jest migający wskaźnik aktualnej linii. Ponadto naciśnięcie dowolnego klawisza potwierdzane jest wzmocnionym dźwiękiem.

Beta-Basic zachowuje sposób wprowadzania instrukcji do pamięci używany przez Basic ZX Spectrum. Wszystkie słowa kluczowe Beta-Basica odpowiadające nowo wprowadzonym instrukcjom są uzyskiwane w trybie graficznym pracy klawiatury (CAPS SHIFT i 9) po naciśnięciu odpowiedniego klawisza. Natomiast dodatkowe funkcje Beta-Basica są wywoływane po wprowadzeniu słowa kluczowego FN i naciśnięciu klawisza odpowiadającego danej funkcji oraz wprowadzeniu znaku „$" dla funkcji tekstowych lub znaku „(" dla funkcji numerycznych.

Jeżeli chce się korzystać z trybu graficznego ZX Spectrum, np.: grafik definiowanych, należy wprowadzić instrukcję KEYWORDS 0. Wówczas instrukcje Beta-Basica nie będą osiągalne z klawiatury. Przywrócenie poprzedniego stanu uzyskuje się po wprowadzeniu w odpowiedni sposób instrukcji KEYWORDS 1.

Wszystkie instrukcje wprowadzane przez Beta-Basic można podzielić na kilka grup. Do pierwszej (najobszerniejszej) należą te, które zwiększają efektywność obliczeń powodując zarazem, że program pisany za ich pomocą jest bardziej czytelny. Niektóre z tej grupy instrukcji wprowadzają do Basica elementy programowania strukturalnego. W języku Beta-Basic podprogram (procedura) może mieć nazwę. Taki podprogram można wywołać podając po słowie kluczowym PROC jego nazwę. Dzięki temu szkieletem każdego programu mogą być wywołania procedur zdefiniowanych przez użytkownika. Niestety, nie można tu definiować procedur z parametrami, co jest typowe dla innych języków, np. dla Pascala. Beta-Basic znacznie rozszerza repertuar dozwolonych postaci instrukcji pętli. Nowe konstrukcje pętli nie wymagają od programisty zadeklarowania zmiennej sterującej, mówiącej o liczbie jej wykonań (jak np. FOR I=1 TO 10). Wyjście z pętli może nastąpić zależnie od wartości warunku.

W zależności od wybranej konstrukcji zakończenie wykonania pętli następuje, gdy warunek nie jest spełniony (DO...WHILE...) lub gdy się spełni (DO... UNTIL...). Możliwe jest także wyjście ze środka cyklu po spełnieniu umieszczonego w wybranym miejscu pętli warunku.

Instrukcja warunkowa IF...THEN została rozszerzona o opcję ELSE. Jeżeli warunek występujący po IF nie będzie spełniony, wówczas nie nastąpi przejście do kolejnej linii programu, a będzie zrealizowana instrukcja występująca bezpośrednio po słowie kluczowym ELSE. Takie konstrukcje instrukcji warunkowych są używane także w wielu językach, jak np.: Algol czy Pascal.

Rozszerzenia interpretera Sinclair Basica objęły także instrukcje skoku oraz skoku do podprogramu. W nowej instrukcji ON...GOTO lub ON...GOSUB można wyszczególnić listę numerów linii. W zależności od wartości wyrażenia, podanego po słowie kluczowym ON wybrany zostanie jeden z wyszczególnianych numerów linii. Do linii o tym numerze nastąpi skok (lub skok do podprogramu).

Warto także wspomnieć o całkiem nowym poleceniu, na ogół nie występującym w językach programowania, a mianowicie o SORT. Umożliwia ono porządkowanie tablic numerycznych, tablic tekstowych i łańcuchów znaków. Sortować można w porządku rosnącym lub malejącym. Dużą zaletą tej instrukcji jest szybkość działania. Tablica tekstowa składająca się z 200 elementów, po 10 znaków każdy, porządkowana jest w czasie około 0,7 s.

Drugą grupą instrukcji Beta-Basicu są tzw. operacje ekranowe. Za pomocą instrukcji z tej grupy można, między innymi, szybko zmieniać atrybuty chwilowe całego ekranu lub dowolnie wybranych jego części. Poza tym narysowane na ekranie powierzchnie ograniczonej dowolnymi krzywymi zamkniętymi mogą być kolorowane. Możliwe jest również przesuwanie naniesionych na ekran kształtów w jednym z czterech podstawowych kierunków. Instrukcje te mogą operować zarówno na całym ekranie, jak i jego częściach. Operacje tego typu są szczególnie przydatne podczas pisania programów edukacyjnych czy gier.

Kolejnym rozszerzeniem jest instrukcja pozwalająca na formatowanie wydruków dowolnych liczb. Każda liczba może zostać wydrukowana zgodnie z określonym po instrukcji PRINT (LPRINT) formatem, bez konieczności odwoływania się do odpowiedniego podprogramu, tak jak to zostało pokazane w p. 3.4.1.

W Beta-Basicu można tworzyć programy samomodyfikujące, co jednak nie jest polecane. Część programu w Basicu może tworzyć łańcuchy znaków, których zawartością będą nowe linie programu wraz z numerami wskazującymi na miejsce, w którym mają być umieszczone. Używając instrukcji wprowadzającej łańcuch do pamięci można tworzyć nowe programy lub modyfikować już istniejące, dopisując nowe linie, czy też zastępując stare nowymi.

Konstrukcja algorytmu oraz zapisanie go za pomocą dostępnych instrukcji języka w pamięci komputera jest pierwszym krokiem tworzenia programu. Drugim, równie istotnym etapem, jest jego uruchomienie. Często w czasie uruchamiania programów przydatne jest śledzenie*) wykonania zleconego komputerowi zadania. Taką możliwość daje Beta-Basic. Prócz śledzenia wykonania programu lub jego fragmentu istnieje możliwość obsługi błędów występujących w programie za pomocą specjalnych programów. W fazie uruchamiania pomocne są instrukcje pozwalające na drukowanie na ekranie lub drukarce wybranych fragmentów lub linii programu, co znacznie ułatwia ich poprawianie.

Ostatnią grupą instrukcji Beta-Basica są rozszerzenia edytora. Można wprowadzać program z automatyczną numeracją linii. Łatwiejszy jest proces poprawiania programu. Poprawianie wybranej linii nie wymaga ustawienia na niej znacznika. Wystarczy podać numer linii jako argument instrukcji EDIT. Poza tym w trybie edycji istnieje możliwość dzielenia złożonych linii programu oraz umieszczenia ich części w dowolnym jego miejscu. Dozwolone jest także łączenie występujących obok siebie linii programu.

Jak już powiedziano, Beta-Basic poza nowymi instrukcjami wprowadza nowe

*) W tym przypadku polega ono na drukowaniu na ekranie numerów aktualnie wykonywanych linii programu.

funkcje. Niektóre z nich dublują już istniejące. Jednakże ich zaletą jest to, że obliczanie ich wartości trwa kilka razy (2÷6) szybciej niż w Basicu ZX Spectrum. Dzieje się tak dzięki zmianie algorytmu ich obliczania lub zmniejszeniu dokładności obliczeń do czterech cyfr znaczących (dotyczy to funkcji trygonometrycznych).

Interpreter Sinclair Basica jest rozszerzony o cztery funkcje konwersji liczb całkowitych z przedziału 0÷65535 (0000÷FFFF) z postaci dziesiętnej na binarną, dziesiętnej na heksadecymalną oraz z heksadecymalnej na dziesiętną. W praktyce dość często wykorzystywana jest funkcja zamieniająca liczby z postaci dziesiętnej na binarną. Jej argumentami są zwykle wartości funkcji logicznej: sumy, iloczynu lub też sumy modulo 2, będących również elementami Beta-Basica.

Bardzo istotnymi funkcjami, które wprowadza omawiany język, są dodatkowe operacje na łańcuchach znaków. Z poziomu Beta-Basica możliwe jest utworzenie łańcucha z zawartości wybranych obszarów pamięci o długości do 65532 znaków. Korzystając z rozszerzonej instrukcji POKE można szybko przemieszczać całe obszary pamięci — podobnie jak w asemblerze Z80.

Poza tym w łańcuchu utworzonym z wybranego obszaru pamięci można poszukiwać innego, określonego wcześniej, łańcucha (nie dłuższego jednakże niż 255 znaków). Funkcja ta bardzo przydaje się przy pisaniu programów edukacyjnych do sprawdzania poprawności udzielanych odpowiedzi. Oprócz tego istnieje możliwość powielania dowolnego łańcucha.

Opisano tu pokrótce tylko wybrane instrukcje i funkcje, o które Beta-Basic rozszerza interpreter Basica ZX Spectrum. Opis przytoczonych tu rozszerzeń nie jest pełny. Nie omówiono tutaj instrukcji i funkcji ułatwiających oszczędne gospodarowanie pamięcią, a nawet w niektórych przypadkach odzyskiwania części pamięci w trakcie pracy programu. Celem tego opisu było przedstawienie Czytelnikowi wybranych rozszerzeń interpretera Basica, które zdaniem autorów mogą być bardzo pomocne przy rozwiązywaniu za pomocą komputera wielu problemów.

Tytułem uzupełnienia należy zasygnalizować nowe instrukcje wprowadzone w wersji 3.0 sprzedawanej od połowy 1985 r.:

— można definiować procedury z parametrami, mające własne zmienne (lokalne), definiowane i dostępne tylko wewnątrz tej procedury,
— procedury mogą wywoływać same siebie, a więc działać w sposób rekursywny,
— ekran może być podzielony na maksimum 128 części (okien) oddzielnie obsługiwanych,
— znaki mogą być dowolnej wielkości (od 1 do 64 znaków w linii),
— istnieje możliwość wpisywania słów kluczowych litera po literze,
— wprowadzono nowe instrukcje do obsługi microdrive'ów.

W sumie 26 nowych i ulepszonych funkcji i około 40 instrukcji. Do programu dołączany jest 80-stronicowy podręcznik.

MegaBasic, powstały w 1984 r., jest niewątpliwie jednym z najlepszych rozszerzeń języka Basic. Jego autorem jest Marc Leaman (program rozprowadza czasopismo *Your Spectrum*).

MegaBasic korzysta z interpretera Basica ZX Spectrum. Dodaje do istniejących instrukcji wiele nowych, głównie graficznych i ułatwiających uruchamianie programów, ale też, jako interpreter nie przyśpiesza bynajmniej ich wykonywania. Został on stworzony po to, by ułatwić użytkownikowi programowanie na poziomie Basica, by nie musiał on fragmentów programu pisać w języku wewnętrznym, bądź używać długich ciągów instrukcji basicowych tylko po to, by np. przesuwać na ekranie obraz z lewej strony na prawą. MegaBasic może być dużą pomocą przy tworzeniu oprogramowania wymagającego dobrej grafiki, a więc programów edukacyjnych, prostych gier, programów demonstracyjnych, ilustrujących wykłady, itp. Może też pomóc mniej wprawnym programistom, szczególnie przy testowaniu napisanego programu (ma dodatkowe funkcje, program-monitor) i wprowadzaniu programu do pamięci (rozbudowany edytor, dyrektywy współpracy z magnetofonem).

Omówienie zmian i rozszerzeń wprowadzanych w MegaBasicu podzielono na kilka części w zależności od funkcji realizowanych przez poszczególne instrukcje.

Edytor

Edytor MegaBasica wprowadza znaczną zmianę w porównaniu z Basicem Sinclaira. Instrukcje i nazwy funkcji nie są słowami kluczowymi wprowadzanymi naciśnięciem pojedynczego klawisza, trzeba je wprowadzać tak jak w innych komputerach, litera po literze. Dla wielu Czytelników może to się wydać wadą (trzeba więcej pisać). Zdaniem autorów stanowi to zaletę — nie trzeba pamiętać, na którym z klawiszy klawiatury umieszczono oznaczenie danej instrukcji. Zresztą, zamiast całej instrukcji można wprowadzać tylko jej skrót zakończony kropką.

W porównaniu ze standardowym edytorem dołączono instrukcje automatycznego numerowania wprowadzonych linii (AUTO), usuwania ciągów linii (DELETE), edycji linii o podanym numerze (EDIT *nr*), instrukcje manipulowania kursorem, kopiowania znaków z jednej linii programu do innej, definiowania ciągów znaków wprowadzanych naciśnięciem pojedynczego klawisza (KEY) (można używać do tego celu klawiszy z górnego rzędu klawiatury). Rozbudowano także edycję znaków w aktualnie analizowanej linii.

Funkcje graficzne

W MegaBasicu wprowadzono możliwość definiowania na ekranie okien. Dzielą one ekran TV na kilka (do 10*⁾) prostokątów o zadanych wymiarach, które mogą

*⁾ Jednocześnie można używać tylko 3 z nich.

być wypełniane treścią niezależnie od siebie. Zdecydowana większość instrukcji graficznych MegaBasica dotyczy tworzenia i wypełniania poszczególnych okien.

W momencie wpisania do pamięci MegaBasica wszystkie okna są nałożone na siebie. Wykonanie instrukcji aktywizującej jedno z okien połączone z podaniem jego numeru (CURRENT) i atrybutów spowoduje jego pojawienie się na ekranie i umożliwi wysłanie do niego informacji za pośrednictwem instrukcji odwołującej się tylko i wyłącznie do tego okna. Cztery okna zdefiniowano á priori:

— zawierające informacje o błędach i linie wprowadzane przez użytkownika,
— automatyczny listing*),
— używane przez instrukcje MegaBasica, np. PRINT czy LIST,
— używane przez monitor MB.

Można mieć zatem na ekranie równocześnie listing wykonywanego programu, jego dane i wyniki.

Każde z okien może być przesuwane do góry lub w dół niezależnie od innych o dowolną liczbę wierszy, a także w lewo lub w prawo (instrukcje SCROLL, PAN). Można definiować różny krój czcionki (FONT; 3 rodzaje**)) i różną wielkość znaków (MODE: od 16 do 64 znaków w linii). Poszczególne znaki mogą być powiększane w zadanej skali (SPRINT), intensywność ich koloru może być regulowana w 16-stopniowej skali (STIPPLE). Wprowadzono także możliwość definiowana i wywoływania sprite'ów (patrz rozdz. 3). Są to znaki o wymiarach 16×16 punktów ekranu mogące poruszać się po ekranie zmieniając swój wygląd i kierunek ruchu. Ich ruch jest sterowany z poziomu asemblera. Jest to podstawowe narzędzie przy tworzeniu gier komputerowych. Służą one do animacji postaci na ekranie.

W MegaBasicu można zdefiniować do 8 takich obiektów, każdy o kilku fazach ruchu oraz o własnych atrybutach (SPRITE). Rozszerzony Basic dopuszcza ponadto instrukcje wymiany zawartości dowolnej części ekranu z pamięcią. Pozwala to na wstępne przygotowanie obrazu i szybkie nałożenie go na aktualnie prezentowany.

Inne instrukcje graficzne MegaBasica (MB) dotyczą definiowania znaków graficznych i zmiany atrybutów ekranu. Ujednolicono układ współrzędnych ekranu. W każdej instrukcji górny lewy róg ekranu ma współrzędne (0,0).

Dodatkowe instrukcje sterujące Basica

W MegaBasicu wprowadzono dodatkowy rodzaj pętli — repeat...until. Konstrukcja ta umożliwia wykonywanie ciągu instrukcji znajdujących się między tymi słowami kluczowymi, dopóki warunek logiczny znajdujący się po słowie until nie będzie spełniony (jego wartość równa się zero). W przeciwnym przypadku

*) Listing, który pojawia się samoczynnie przed uruchomieniem programu lub po jego zatrzymaniu w momencie naciśnięcia klawisza ENTER.

**) Zdefiniowany instrukcją FONT krój czcionki obowiązuje we wszystkich oknach.

zostanie wykonana następna instrukcja po until (jest ona podobna do DO...UNTIL z języka Beta-Basic). W MegaBasicu można umieszczać jedna w drugiej do 10 takich pętli.

Rozszerzony Basic zawiera także instrukcje definiowania bardzo wygodnego narzędzia programowania, a mianowicie procedur. Linia, której pierwszym znakiem jest C definiuje początek procedury o podanej nazwie i parametrach. Linia o postaci ENDPROC_ *nazwa* kończy definicję. Taki ciąg instrukcji może być wywołany z programu głównego przez podanie nazwy i parametrów — o ile to wygodniejsze niż GOSUB! Ułatwiono także współpracę z częściami programu napisanymi w języku wewnętrznym. Poprzez stos można przekazywać parametry do podprogramu asemblerowego (CALL).

W MegaBasicu wprowadzono także (w postaci bardzo uproszczonej) mechanizm używany w znacznie większych niż Spectrum komputerach — wielozadaniowość. W tej wersji polega to na umożliwieniu wykonywania na przemian instrukcji z dwóch różnych miejsc programu.

MegaBasic zawiera także instrukcje operacji na stosie Basica — wpisywania i pobierania informacji, zerowania wskaźnika stosu. Można także przesyłać dwubajtowe wartości do pamięci (DOKE).

Instrukcje ułatwiające uruchamianie programu

Twórca MegaBasica M. Leaman wprowadził możliwość wywoływania po zakończeniu każdej wykonywanej linii programu dodatkowej procedury, np. drukującej wartości zmiennych (BRANCH). Można także śledzić kolejność wykonywania linii i deklarować prędkość działania programu (TRON, TROFF, SPEED).

Działający program można przerwać wciskając klawisz spacji i jeden z trzech klawiszy F, E, R: E — powoduje przejście do edytora, R — powtórny start systemu (nie niszczy się treści programu), F — przejście do programu monitora*).

Wprowadzono także możliwość blokowania i odblokowywania klawisza BREAK.

Dźwięk

MegaBasic ma dodatkowy generator dźwięku, który działa także w trakcie wykonywania innych instrukcji programu. Liczby opisujące czas trwania, wysokość dźwięku i liczbę powtórzeń są wpisywane do bufora. Jego zawartość jest co 1/50 s sprawdzana i na podstawie odczytanych wartości jest generowany dźwięk. Wprowadzono możliwość generowania szumów i rozszerzono możliwości instrukcji BEEP.

* * *

*) Monitor jest to program umożliwiający podejrzenie tego, co naprawdę jest zapisane w poszczególnych komórkach pamięci i rejestrach procesora. Pozwala na zmianę zawartości rejestrów i pamięci, testowanie programów asemblerowych, itd. Jest to podstawowe narzędzie potrzebne przy uruchamianiu programów napisanych w języku wewnętrznym (patrz także rozdz. 8.10).

Z innych dodatkowych względem Basica ZX Spectrum instrukcji należy wymienić rozkazy dotyczące kopiowania programów i katalogowania zbiorów zapisanych na taśmie, a także instrukcje umożliwiające przesyłanie znaków w kodzie ASCII zamiast na ekran do dowolnego portu za pośrednictwem napisanego przez użytkownika programu asemblerowego.

Ten z konieczności pobieżny opis MegaBasica przekonuje o tym, że zawiera on poważną liczbę nowych instrukcji, które powinny skrócić czas opracowania i testowania programu, zapewniając jednocześnie znacznie atrakcyjniejszą formę prezentacji jego wyników.

Aby obiektywnie ocenić system należy podać jeszcze jego wady. Podstawową jest znaczne ograniczenie miejsca w pamięci na program użytkownika. Może zająć on do 21 kB, co przy dużych zbiorach danych, licznych zmianach zawartości ekranu może okazać się niewystarczające. Korzystanie z MegaBasica utrudnia skutecznie wyjątkowo źle napisana instrukcja, bardzo pobieżnie i niedokładnie traktująca opisy działania i parametry poszczególnych instrukcji. Trudno jest przyzwyczaić się do nowej składni Basica — parametry nowych instrukcji wymagają połączenia ze słowem kluczowym znakiem"_" (podkreślenie), np. FADE_1, w przeciwieństwie do dotychczasowych np. PLOT 1,1. Pewne funkcje utworzono na wyrost. Przy dość małej rozdzielczości grafiki ZX Spectrum zawartość okien na ekranie jest mało czytelna. Wielozadaniowość to także zbyt dużo jak na ograniczone możliwości tego mikrokomputera.

Autorzy mieli ponadto pewne trudności z uruchamianiem poprawnego działania niektórych instrukcji np. dotyczących dodatkowej metody generacji dźwięku.

MegaBasic ma też błędy. Niekiedy nieoczekiwanie wykonuje RESTART 0 lub zawiesza się.

W sumie MegaBasic jest niezłym narzędziem do przygotowywania niezbyt dużych programów o bogatej oprawie graficznej. Niestety, nie pokuszono się o przyśpieszenie działania niektórych standardowych instrukcji Basica, co mogłoby znacznie zwiększyć zainteresowanie tym językiem.

* Kompilatory języka Basic 8.4

Kompilatory a interpretery 8.4.1

Jak już wielokrotnie wspominano Basic ZX Spectrum jest interpretowany przez program zawarty w pamięci stałej komputera. Jego działanie polega, mówiąc w skrócie, na pobieraniu kolejnych instrukcji Basica, rozpatrywaniu ich znaczenia, a następnie wywoływaniu procedur realizujących zadawane rozkazy i funkcje oraz wykonywaniu ich. Cały ten proces powtarzany jest przy wykonaniu każdej z instrukcji programu użytkownika. Jak można zauważyć, samo wykonanie stanowi tylko ułamek czasu poświęconego na wyszukiwanie i „zrozumienie"

polecenia zapisanego w Basicu. Taki sposób wykonywania programu ma zatem jedną zasadniczą wadę — szybkość jest mała, jego działanie jest nieefektywne.

Trudno jest znaleźć miarę efektywności służącą do porównywania programów realizujących zadany algorytm. Na ogół używa się dwu kryteriów — szybkości działania i liczby bajtów pamięci zajętych przez program, zwykle jednak czynnikiem decydującym o tym, który program jest uważany za lepszy, jest czas jego wykonania. Wynikła stąd konieczność stosowania takich metod pisania programów, aby działały one jak najprędzej, ale jednocześnie proces pisania był łatwy. Dlatego przy tworzeniu dużych programów unika się programowania w asemblerze, a używa się języków wysokiego poziomu, w sposób efektywny tłumaczonych na kod maszynowy, a równocześnie dających możliwość wygodnego formowania rozwiązań problemów, ich poprawek itd.

Interpretery są bardzo użyteczne w trakcie tworzenia programu — działanie programu można przetestować natychmiast po jego wprowadzeniu. Niestety, wymóg efektywności czasowej nie jest przez nie spełniony. Wobec tego wymyślono inny sposób tłumaczenia programu na kod wewnętrzny komputera — k o m p i l a c j ę. Polega ona na tym, że najpierw kompletuje się cały program, następnie pobiera się instrukcję po instrukcji, znajduje ich sens i zamienia je na ciąg rozkazów procesora. Na tym kończy się tłumaczenię. Jego wynikiem jest zapisanie całego programu użytkownika w postaci programu maszynowego. I dopiero wtedy, oddzielnym poleceniem, uruchamia się jego działanie. Podczas wykonania programu zamiast trzech czynności wykonuje się jedną. I to tę trwającą zwykle najkrócej. To przyśpieszenie działania odbywa się kosztem wygody użytkownika — wprowadzenie choć jednej zmiany w tekście programu powoduje konieczność ponownego tłumaczenia całości. Na przetłumaczenie trzeba czekać i to często dość długo (nawet kilkanaście minut).

Metody kompilacji programu są różne — mniej lub bardziej złożone, dające szybszy lub wolniejszy kod wynikowy. W przypadku Basica stosuje się tłumaczenie bądź do kodu maszynowego, bądź do pewnego języka pośredniego (zwanego z ang. *p-code'm*). Zastosowanie tej drugiej metody daje wolniej działający kod (ten zapis w języku pośrednim jest w trakcie wykonania interpretowany), ale jest on znacznie krótszy (nawet niż odpowiadający mu program maszynowy). Sam kompilator ma prostszą, mniej złożoną budowę w porównaniu z programem tłumaczącym do kodu wewnętrznego procesora. Z kolei tłumaczenie bezpośrednio do kodu maszynowego skraca czas wykonania, ale zapis jest dłuższy, zwykle dłuższy niż odpowiadający mu zapis w Basicu.

Zwykle do kodu otrzymanego w wyniku translacji kompilator dołącza pewne bloki procedur wspomagające wykonanie programu (tzw. procedury *run-time*). Zawierają one podprogramy współpracy z urządzeniami zewnętrznymi (klawiatura, ekran), funkcje matematyczne (np.: sinus, cosinus, mnożenie) i inne.

Istnieją kompilatory, które nie tylko tłumaczą program na język wewnętrzny maszyny, ale również potrafią ten kod optymalizować, niejako poprawiać program użytkownika, np. zastępując wielokrotne wyliczanie pewnej wartości raz obliczonym wyrażeniem.

Jak stwierdzono, Basic ZX Spectrum działa stosunkowo wolno, ale jest w zamian wygodny w użyciu. Warto zatem programy często używane, sprawdzone już i uruchomione, przetłumaczyć na język wewnętrzny, by działały one szybciej, by czas oczekiwania na wyniki obliczeń skrócić kilkakrotnie. Niestety, choć sprzedawanych jest wiele kompilatorów Basica dla ZX Spectrum właściwie każdy z nich ma bardzo poważne ograniczenia co do rodzaju tłumaczonych instrukcji. Ze znanych autorom, jedynie BLAST firmy OCCP tłumaczy pełny zestaw instrukcji Basica ZX Spectrum, ale za to zajmuje prawie całą pamięć komputera.

Kompilatory Basica ZX Spectrum można podzielić na dwie grupy: realizujące operacje stałoprzecinkowe i realizujące operacje zmiennoprzecinkowe. Translatory pierwszej grupy traktują wszystkie liczby występujące w programie jako całkowite z zakresu (−32768; +32767), a więc takie, które dają się zapisać na 16 bitach. W związku z tym zmienne liczbowe są zapamiętywane na dwu bajtach, a nie na pięciu jak w Basicu ZX Spectrum. Obsługują one tylko niektóre z instrukcji Basica, a także tylko wybrane funkcje. Na ogół realizowane są instrukcje arytmetyczno-logiczne i operacje na ciągach znaków. Tylko niektóre z tych translatorów dopuszczają użycie tablic, a jeżeli nawet, to tylko jednowymiarowych i to często tylko jednej w programie. W zamian wprowadzane są pewne dodatkowe instrukcje, na ogół umieszczane w linii komentarza REM. Pozwalają one na uzupełnienie programu o wstawki w asemblerze, blokowanie i odblokowywanie klawisza BREAK itp. W sumie, są to bardzo ograniczone podzbiory Basica, sprowadzają go one do nieco lepszego asemblera i mają niewielkie zastosowanie praktyczne (do gier, programów graficznych). Działają za to bardzo szybko, kilkadziesiąt razy szybciej niż pierwowzór. Programy tłumaczące tego typu zajmują niewiele miejsca w pamięci (ok. 5 kB). Program jest tłumaczony do kodu maszynowego. Przetłumaczony program (tzw. kod wynikowy) jest umieszczany w obszarze pamięci o wysokich adresach, na ogół powyżej RAMTOP. Przykładami takich translatorów są: MCODER, IS Compiler, Super Compiler firmy Softek. Dobry translator stałoprzecinkowy, bez licznych ograniczeń, byłby na pewno przydatny w wielu programach użytkowych, lecz jak dotychczas nie ma takiego programu.

Translatory, które mogą operować na liczbach zmiennoprzecinkowych są bardziej skomplikowane, obsługują też więcej instrukcji Basica ZX Spectrum. Dopuszczają one używanie standardowych funkcji Basica, tablic (jednowymiarowych). Zajmują one nieco więcej pamięci (ponad 6 kB), ale przede wszystkim działają wolniej niż opisywane poprzednio. Program przetłumaczony takim translatorem działa co najwyżej kilka razy szybciej niż interpretowany. Dość szybki i wygodny w użyciu jest np. FP V1.7 firmy Softek.

Przetłumaczony program może zostać zapamiętany na taśmie magnetofonowej. Aby następnie wczytany program mógł działać razem z jego kodem, na taśmę musi zostać wysłana biblioteka procedur obsługi wykonania (ang. *run-time*). Sam proces tłumaczenia jest dosyć szybki.

W tym podziale nie mieści się aktualnie najlepszy z kompilatorów Basica ZX Spectrum — BLAST firmy OCCP. Pojawił się w 1985 r. i stał się jedną z sensacji targów komputerowych w Londynie. Jest to jedyny dostępny pełny kompilator Basica ZX Spectrum, może przetłumaczyć na kod wewnętrzny wszystkie jego instrukcje. Mało tego, potrafi optymalizować produkt końcowy, może tłumaczyć do kodu pośredniego, bądź do kodu maszynowego (szybszy, ale dłuższy). W zestawie BLAST dołączony jest też program TOOLKIT rozszerzający możliwość edycji i testowania programu w Basicu. BLAST ma nowe instrukcje Basica, np. nowe pętle (repeat... unitil, while...wend), funkcje definiowane przez użytkownika złożone z wielu linii. Są one wykonywane znacznie szybciej niż standardowe instrukcje Spectrum.

Te możliwości zostały okupione bardzo obszernym kodem samego programu tłumaczącego. Zajmuje on prawie całą pamięć komputera, pozostawiając na program użytkownika nieco ponad 2700 bajtów. To bardzo mało, ale na szczęście używając programu TOOLKIT można dłuższe programy dzielić na części akceptowane przez translator, zapamiętywać na taśmie i następnie częściami tłumaczyć, zapisując wynik na taśmie. W trakcie wykonania w pamięci musi tylko pozostać (oprócz programu) zespół procedur *run-time* (ok. 5 kB). Przetłumaczony i wczytany z taśmy program jest ładowany w pole programu źródłowego i uruchamia się go poleceniem RUN. Dużą niedogodnością jest pozostawienie tylko 5 s na zmianę taśm podczas translacji z taśmy na taśmę. Jeżeli w tym czasie nie założy się na magnetofon kasety, na której ma zostać nagrany przetłumaczony program, to nie zostanie on zapamiętany. BLAST staje się naprawdę użyteczny dopiero przy zastosowaniu microdrive'ów.

Niech użytkownika nie zdziwią pojawiające się na ekranie przypadkowe punkty i paski — BLAST w trakcie tłumaczenia programu zajmuje także pamięć ekranu.

Kompilator ten jest w nietypowy sposób zabezpieczony przed skopiowaniem. Na firmowym opakowaniu zamieszczona jest czterokolorowa tabela (czerwony, biały, żółty, zielony) o wymiarach 26 × 40 pól. Po wczytaniu BLASTA do pamięci komputera, użytkownik musi określić kolory czterech losowo wybranych miejsc tabelki. Błędna odpowiedź, bądź wciśnięcie niewłaściwego klawisza powoduje wyzerowanie systemu. Sama kompilacja trwa stosunkowo długo.

Autorzy mieli możność testowania jednej z pierwszych wersji BLAST Basica. Niestety zawierała ona jeszcze błędy, niektóre z jego instrukcji nie działały tak jak powinny. Według opinii firmy NOWATECH -. Katowic aktualne wersje działają całkiem poprawnie.

Kompilatory Basica na ZX Spectrum wymagają jeszcze trochę dopracowania. Mają one zwykle bardzo ograniczone możliwości, bądź jeżeli tłumaczą pełny zestaw instrukcji Spectrum zajmują prawie całą pamięć, co pozwala tłumaczyć tylko dość krótkie fragmenty kodu.

Kompilatory Basica mogą być przydatne przy tworzeniu gier komputerowych, ale też i w programach obliczeniowych, ze względu na dużą precyzję działań zmiennoprzecinkowych. W wielu przypadkach lepiej jest jednak zapisać od nowa

algorytm w innym języku np.: Pascalu czy C. Języki te są bardziej eleganckie, szybsze i przyjemniejsze w użyciu niż Basic.

* * Logo 8.5

Jeszcze w latach 60-tych przewidując nieuchronność wprowadzenia informatyki do szkół, rozpoczęto w USA badania nad najodpowiedniejszym do tego celu językiem programowania. W wyniku prac przeprowadzonych pod kierunkiem S. Paperta w Artificial Inteligence Laboratory — Massechussets Institute of Technology (MIT) — powstała obecna wersja Logo. Wraz z dynamicznym rozwojem mikrokomputerów w latach osiemdziesiątych powstało wiele implementacji tego języka na poszczególne „maszyny". Logo na ZX Spectrum, uznane przez organizację normującą poszczególne rozwiązania — Logo Computer Systems Incorporation (LCSI), powstało w 1984 r. i ukazało się na rynku pod nazwą Sinclair Logo LCSI SOLI.

Logo jest językiem interpretowanym. Implementacja tego języka wymaga kilkudziesięciu kilobajtów pamięci, gdyż interpreter zajmuje około 25 kB, oraz grafiki dużej rozdzielczości. Język ten z dnia na dzień zyskuje coraz więcej zwolenników, zdarzają się jednak także przeciwnicy. O popularności i dużej sile ekspansji tego języka może świadczyć fakt, że umieszcza się alternatywnie do Basica interpreter tego języka w pamięci ROM, np. Atari 520ST. Poza tym, obecnie program interpretera Logo jest jednym z pierwszych, który można kupić jako oprogramowanie dodatkowe każdego komputera osobistego.

Powstają więc pytania: Co to za język? Czym zdobywa sobie tak dużą popularność?

Logo jest uniwersalnym, w pełni konwersacyjnym językiem programowania. Ma on najwięcej wspólnych cech z językiem LISP (od ang. *LIST Processor*), np. struktury danych. Charakterystyczną cechą Logo wykorzystywaną w edukacji — szczególnie w nauczaniu geometrii — jest bardzo rozbudowana grafika. Należy tu nadmienić, że pierwsze implementacje Logo nie miały takich możliwości graficznych.

W związku z dużymi możliwościami operowania na strukturach listowych (patrz dalej) za pomocą Logo można swobodnie wykonywać dowolne operacje na tekstach. W języku tym możliwe jest definiowanie zmiennych globalnych i lokalnych, istnieje iteracja i rekurencja oraz, jak już wspomniano, bardzo dobra kolorowa grafika. Można powiedzieć, że programowanie w Logo jest dla programisty połączeniem Pascal-owskiego podejścia proceduralnego z Lisp-owymi strukturami danych z możliwością pełnego zobrazowania graficznego na ekranie telewizora czy też drukarce. Praca z programem jest konwersacyjna i bardzo wygodna, można np. wyświetlić na ekranie nazwy zdefiniowanych procedur wraz z ich parametrami lub treść dowolnej procedury (procedur).

Gramatyka Logo jest stosunkowo prosta. Każdy program jest zbiorem procedur i funkcji. Instrukcje traktowane są jako wywołania procedur podstawowych

(ang. *primitives*). Realizacja w trybie bezpośrednim jest identyczna jak w trybie programowym. Nie ma tu numeracji linii programu oraz rozbudowanej, tak jak w Basicu, interpunkcji w instrukcjach we/wy. Podstawowym separatorem jest odstęp (ang. *space*), o czym trzeba zawsze pamiętać.

W fazie tworzenia programu odczuwa się wiele zalet efektywnego edytora ekranowego Logo. Można w nim pisać i poprawiać całe procedury. Istnieje możliwość redagowania kilku procedur jednocześnie lub też zdefiniowanych zmiennych globalnych w czasie edycji. Zawartość edytowanego programu przegląda się poruszając kursorem, który może być przesuwany znak po znaku, linia po linii lub strona (ekran) po stronie, w dowolnym kierunku. Można też kasować pojedyncze znaki, a nawet całe linie. Wniesione do redagowanej procedury poprawki można usunąć, o ile procedura ta nie została ponownie zdefiniowana. Mimo że praca z edytorem Basica na ZX Spectrum jest wygodna, to o wiele przyjemniejsza jest praca z edytorem Logo.

Dużą zaletą tego języka jest to, że dzięki swojej strukturze utrudnia on chaotyczne programowanie, choć jak mówi przysłowie „dla chcącego nic trudnego". Każdy złożony problem powinien być rozbijany na problemy składowe (problemy podstawowe), a te winny być rozwiązywane za pomocą odpowiednio skonstruowanych procedur. Można je definiować niemal w dowolnym momencie pracy komputera. Taki zapis algorytmu powoduje to, że jest on czytelny i przejrzysty. Kolejną zaletą programowania w Logo jest to, że w czasie rozwiązywania problemu nie jest wymagana dobra znajomość systemu operacyjnego mikrokomputera, w przeciwieństwie do Basica, w którym każdy poważniejszy program zawiera tu i ówdzie tajemnicze POKE lub PEEK. Poza tym programy w Logo mogą być łączone z programami pisanymi w asemblerze Z80.

Logo ma również wady, wśród których należy wymienić dużą liczbę instrukcji czyli procedur podstawowych. Jest ich około 150. Jednym z powodów tak dużej liczby instrukcji jest zapewnienie obsługi urządzeń zewnętrznych z poziomu Logo. Między innymi implementacja ta zawiera cały zbiór instrukcji pozwalających na współpracę z pamięcią zewnętrzną (magnetofonem), drukarką lub specjalnym robotem, mogącym poruszać się np. po podłodze, którego ruchy mogą być obrazowane na ekranie za pomocą specjalnego znacznika, oraz innymi urządzeniami zewnętrznymi. Logo na ZX Spectrum ma kilkanaście (jak do tej pory zauważonych) błędów implementacji*). Są to jednak błędy popełnione w fazie pisania interpretera i można je wyeliminować.

*) „Błędem" znacznie utrudniającym pracę jest zbyt krótka przerwa czasowa pomiędzy poszczególnymi segmentami programu w Logo zapisywanymi na taśmie magnetofonowej. Jest to bardzo uciążliwe w czasie wczytywania zapisanych na taśmie programów do pamięci komputera. Programy te nie chcą się wczytać. Po to, by temu zaradzić należy po wczytaniu interpretera Logo do pamięci ZX Spectrum wykonać polecenie:

z poziomu Logo: DEPOSIT 34624 80

lub

z poziomu języka Basic: POKE 34624, 80

Wszystkie instrukcje Logo można podzielić na kilka grup, a mianowicie:

— instrukcje graficzne — pozwalają na kierowanie ruchami żółwia (czyli znacznika zdefiniowanego na ekranie) zostawiającego lub nie — zgodnie z życzeniem programisty — za sobą ślad. Określają one aktualne atrybuty na ekranie, takie jak: kolor tła, kolor atramentu, jasność czy też błyskanie, inwersja itd. (podobnie jak w Basicu), instrukcje definiujące aktualne położenie żółwia czy też aktualny kierunek jego ruchu,
— instrukcje definiujące zmienne. W Logo istnieje możliwość tworzenia zmiennych lokalnych określonych w obręcie danej procedury oraz zmiennych globalnych określonych we wszystkich zdefiniowanych procedurach. Można nadawać tym zmiennym wartość oraz odwoływać się do nich przez odsyłacze,
— instrukcje arytmetyczne i logiczne — są to funkcje arytmetyczne oraz podstawowe operatory arytmetyczne. Czasem odczuwa się w Logo brak funkcji logarytmu, ale można dopisać ją do interpretera samemu, tak jak w niejednej z krążących po naszym kraju wersji tego języka na ZX Spectrum. Poza tym są jeszcze operatory logiczne i predykaty**) — oznaczone w tej implementacji literą „P" na końcu słowa kluczowego (w innych wersjach „P" jest zastępowane przez „?"),
— instrukcje sterujące (warunkowe, iteracyjne itp),: instrukcja warunkowa typu IF...THEN...ELSE, instrukcja powtórzenia wykonania danej sekwencji, nadawanie wartości procedurom, instrukcja wyjścia z poziomu realizowanej procedury, instrukcje wyjścia z programu itd.,
— operacje na listach — pozwalają one na tworzenie list poprzez dołączenie elementów do listy już istniejącej. Listą taką może być również lista pusta. Poza tym możliwe jest pobieranie elementu z danej listy, obliczanie wartości wyrażeń będących listami, itp.
— instrukcje wejścia-wyjścia — dotyczą obsługi magnetofonu, drukarki, monitora, łącza w standardzie RS 232 i innych urządzeń zewnętrznych,
— instrukcje definiowania i redefiniowania procedur — pozwalają na zmiany nazw procedur; pisanie procedur, które definiują inne procedury itd. Za pomocą tych instrukcji dozwolone jest pisanie programów samomodyfikujących się,
— instrukcje systemowe — umożliwiają operacje na poszczególnych bajtach pamięci, łączenie programów w Logo z programami w kodzie maszynowym itp.
— instrukcje edytora — pokrótce omówione wcześniej.

Następnie poprawiony program trzeba ponownie nagrać na taśmę (w miejsce starego). Np. po przejściu z Logo do Basica poleceniem BYE robi się to w następujący sposób:

```
SAVE "Logo" CODE 24832, 25000
```

(informacja udzielona przez p. Jarosława Kanię).

**) Są to stosowane w logice funkcje zdaniowe przyjmujące wartości: prawda lub fałsz.

Należy podkreślić, że bardzo ważnym atutem Logo jest możliwość tworzenia własnego słownika instrukcji. W ten sposób można stworzyć własny język komunikowania się z komputerem. Za pomocą Logo daje się w dość prosty sposób napisać nawet interpreter wymyślonego przez siebie języka. Należy jednak pamiętać, że interpreter napisany w ten właśnie sposób byłby bardzo wolny, gdyż interpreter Logo jest mało efektywny. Niemniej napisanie programu interpretera miałoby niewątpliwie walory edukacyjne.

Większość kursów Logo rozpoczyna się od nauki tzw. grafiki żółwia (ang. *turtle graphics*). W początkowym etapie uczniowie poznają możliwości graficzne tego języka. Stąd też u wielu osób, które tylko co nieco słyszały o Logo utarło się przekonanie, że grafika jest podstawową siłą tego języka. Na pewno tak nie jest, gdyż grafika tylko pomaga w zobrazowaniu wszelkich poczynań użytkownika, chociaż biorąc pod uwagę charakter tego języka jest ona również bardzo pouczająca. Za pomocą prostych instrukcji graficznych można uczyć się dobrego stylu programowania oraz filozofii Logo.

Najistotniejszą częścią Logo są jednak operacje na strukturach listowych, zapisywanych jako zbiory obiektów ujętych w nawiasy kwadratowe. Elementami listy mogą być ciągi znaków oddzielone od siebie odstępami lub inne listy, np. listą jest [[1 2] Ewa Stasia Kasia Emilka 1]. W Logo istnieje także pojęcie listy pustej, którą oznacza się przez [ ]. Podstawowymi operacjami na listach są:

— pobieranie z listy pierwszego lub ostatniego elementu,

— odłączanie od listy pierwszego lub ostatniego elementu.

Poza tym istnieje możliwość tworzenia struktur listowych przez dołączenie obiektu na początek lub koniec listy. Wymienione operacje podstawowe uzupełniają predykaty (funkcje zdaniowe), których wartością jest prawda lub fałsz, sprawdzające czy dana struktura jest listą, czy dwie listy są sobie równe, czy dany obiekt jest elementem danej listy lub też, czy dana lista jest listą pustą. Łącząc operacje na listach z rekurencją można w elegancki i prosty sposób rozwiązać wiele problemów. Poza tym wyjaśnienie samej rekurencji na podstawie operacji na strukturach listowych jest pełniejsze i bardziej przekonywujące niż w oparciu o grafikę żółwia.

W Logo — jak już zostało wspomniane — istnieje możliwość pisania programów samomodyfikujących się, czyli programów, które w czasie swojego działania zmieniają swoją treść. Do słowa, które ma być nazwą procedury można przypisać listę, która jest treścią procedury. Do tego celu używa się instrukcji pozwalającej zamienić tekst procedury na listę, którą można następnie poddać odpowiedniej obróbce, wykorzystując operacje na strukturach listowych.

Można mieć nadzieję, że po przeczytaniu krótkiego opisu języka Logo Czytelnik będzie przekonany, iż jest to jeden z lepszych języków „pierwszego kroku". Daje on przegląd prawie całej metodologii programowania. Uczy algorytmicznego myślenia, precyzyjnego formułowania problemów, dzielenia problemów złożonych na proste, itd. Logo nie jest tylko językiem programowania, reprezentuje także pewną filozofię uczenia się.

Dlatego właśnie ten język został wybrany przez Polskie Towarzystwo Informatyczne, które było konsultantem Ministerstwa Oświaty i Wychowania, jako język podstawowy do nauczania przedmiotu „elementy informatyki" w szkołach średnich. W związku z tym została już uzgodniona polska terminologia — nazwy procedur podstawowych języka Logo. Ze względu na ubogą fleksję języka angielskiego było to zadanie bardzo trudne. Nie jesteśmy pierwszym krajem, który wykonał tę pracę, przed nami zrobili to Francuzi, Niemcy, a nawet Bułgarzy.

Być może w niedługim czasie powstanie prawdziwie polski interpreter Logo, bardziej efektywny oraz pozbawiony tych błędów co interpreter Sinclair Logo. Bliższe informacje o Logo można znaleźć w pracach [ ].

* * Forth 8.6

Cechy języka 8.6.1

Język Forth jest dziełem R. H. Moore'a, pracownika obserwatorium astronomicznego, który pod koniec lat sześćdziesiątych stwierdził, że posiadane przez niego narzędzia informatyczne są zbyt słabe, aby szybko i efektywnie prowadzić konieczne w jego pracy obliczenia. Forth przez parę lat był mało znany, jednak powstanie mikrokomputerów przyczyniło się do znacznego zwiększenia zainteresowania tym językiem.

Dlaczego? Przyczyn jest kilka.

Po pierwsze, programy napisane w tym języku wykonują się bardzo szybko, niewiele wolniej niż w asemblerze (szacunkowo o 20% do 75%). Podstawową metodą tłumaczenia programów na kod maszynowy jest interpretacja, a więc metoda, która na ogół uważana jest za wolną. Ponieważ jednak w trakcie wstępnej redakcji programu używa się metody zbliżonej do kompilacji (patrz opis kompilatorów Basica), a także końcowy zapis programu ma specjalną postać, jego interpretacja w fazie wykonania następuje bardzo szybko.

Po drugie, ze względu na ową postać zapisu program zajmuje w pamięci bardzo mało miejsca, mniej niż odpowiadający mu program asemblerowy (!). Ta metoda zapisu wynika ze struktury samego języka. Odpowiednikiem instrukcji innych języków są tzw. słowa, nazwy z przyporządkowanymi im ciągami działań. Na podstawie już zdefiniowanych słów definiuje się następne, na ich podstawie kolejne, itd. Proces definiowania słowa nazywa się kompilacją (semikompilacją). W szczególności, cały program w języku Forth jest jednym słowem. Zatem program taki można zapisać jako ciąg wywołań innych słów; tj. ciąg wywołań podprogramów, a więc ciąg instrukcji CALL. Ponieważ wiadomo, że chodzi o wywołanie podprogramów, zatem wystarczy zapisać tylko ciąg adresów tych podprogramów. Taki sposób zapisu (w uproszczeniu) nazywa się kodem nizanym. Dzięki zastosowaniu go końcowy program jest zwarty.

Programy tłumaczące Forth na język wewnętrzny nie są zbyt skomplikowane, nie potrzebują także wiele pamięci.

Jak z tego wynika, Forth warto stosować do pisania programów działających bardzo szybko — głównie obliczeniowych, ale też graficznych, do sterowania urządzeń i to w komputerach niekoniecznie z dużą pamięcią. Wymaga on jednak od użytkownika o wiele więcej inwencji i staranności w pisaniu niż inne, omówione tu, języki programowania.

Wszystkie obliczenia w Forthcie zapisywane są w Odwrotnej Notacji Polskiej (ONP). Ta beznawiasowa notacja wymaga podawania najpierw argumentów, a potem znaku działania, np. 5 7 + 3 — jest równoważne (5 + 7) − 3. Wprowadzenie tej notacji wynika z kolejnej cechy języka Forth — wszystkie działania prowadzone są na stosie. Pojęcie stosu było już wyjaśnione (rozdział 4) — w danej chwili dostępne są tylko elementy znajdujące się na jego szczycie. Wszelkie operacje dotyczą zatem argumentów wpisanych na szczyt stosu. W podanym przykładzie:

```
→7           →3
5 +  →12     12 -  →9
```

strzałką oznaczono aktualny szczyt stosu. Na stosie mogą być przechowywane dowolne argumenty. Ich interpretacja zależy od rodzaju wykonywanej operacji. Z definicji, Forth operuje na dwóch stosach: danych i adresów podprogramów. Zawartość tych stosów można ze sobą wymieniać. Użytkownik może także zdefiniować własny stos roboczy.

Lista instrukcji tego języka zależy od zdefiniowanego słownika. Na świecie operuje się dwoma jego standardami — tzw. fig-Forth i Forth-79. Oba są stosowane na mikrokomputerach i zawierają bibliotekę zdefiniowanych á priori najpotrzebniejszych słów. Ten standardowy słownik obejmuje operacje arytmetyczno-logiczne na dwu-, trzy- i czterobajtowych liczbach, operacje wymiany zawartości stosów, pętle (*begin...until*, *begin...while...repeat*, *do...loop*), instrukcje warunkowe (*...if...else...endif*). Na ogół w języku Forth nie prowadzi się obliczeń zmiennoprzecinkowych. Do translatora tego języka często dołączany jest dobry asembler, ze względu na możliwość umieszczania w programie wstawek w języku wewnętrznym.

Obecnie Forth zdobył sobie uznanie w dwu dziedzinach zastosowań. Po pierwsze, w robotyce do szybkiego i precyzyjnego sterowania manipulatorami. Po drugie, przy uruchamianiu układów logicznych — jako język opisujący sposób analizy pracy mikrokomputera. Trwają próby stworzenia specjalizowanych układów sprzętowych realizujących podstawowe instrukcje tego języka. Taki mikrokomputer byłby bardzo szybki, a więc nadawałby się do obliczeń inżynierskich i do sterowania.

Należy jednak z całą stanowczością stwierdzić, że jest to język trudny, na pewno nie nadający się do celów dydaktycznych. W trakcie pisania programu trzeba pamiętać o wielu niuansach zapisu — każda spacja jest traktowana jako

separator, przy zakończeniu pętli zmienna sterująca musi znajdować się na szczycie stosu, wiele znaków (np.@, !, .) ma specjalne znaczenie, a oprócz tego wszystkie operacje zapisywane są w odwrotnej notacji polskiej. Sprawia to, że program jest bardzo nieczytelny, choć na ogół krótki. Uruchamianie go utrudnia uboga diagnostyka błędów. Reasumując, na pewno algorytm zapisany w języku Forth będzie działał bardzo szybko, ale zapisać go jest bardzo trudno. Opis języka Forth zamieszczono w [26] oraz numerach *Informatyki* z 1984 r.

Opis implementacji 8.6.2

Forth doczekał się co najmniej kilku realizacji na ZX Spectrum, np. Advanced Spectrum Forth (Melbourne House), Abersoft Forth (Abersoft) i FP50 (CP Software). Dwa pierwsze, to implementacja standardu fig-Forth, trzecia — to Forth zmiennoprzecinkowy.

Wszystkie rozkazy wprowadza się jako ciągi znaków, a nie przez słowa kluczowe zapisane na jednym klawiszu. Liczby stałoprzecinkowe w Forthcie mogą być jedno-, dwu- i czterobajtowe. Ponieważ wprowadzono operacje dzielenia modulo *n*, liczby po przeskalowaniu mogą być traktowane jako ułamkowe (stałoprzecinkowe). Pozwala to na wykonywanie działań na liczbach ułamkowych bez potrzeby definiowania formatu zmiennoprzecinkowego. Operacje stałoprzecinkowe zawsze wykonują się o wiele szybciej.

Fig-Forth używa części pamięci ZX Spectrum jako pamięć zewnętrzną (tzw. RAM-disk). Pamięć ta jest podzielona na 12 bloków, tzw. ekranów, zawierających po 1024 znaki. Służą one do zapamiętywania programów uruchomianych z użyciem wewnętrznego edytora. Ich zawartość można przesyłać na taśmę. Nowe, utworzone przez użytkownika słowa mogą być dołączone do standardowego słownika. Fig-Forth nie ma wielu funkcji arytmetycznych, które ma Basic, a które operują formatem zmiennoprzecinkowym (np.: SQR, SIN). Dołączono natomiast wiele słów pozwalających korzystać z możliwości graficznych (PLOT, DRAW, UDG, INK, PAPER itd.) generacji dźwięku (BEEP) i współpracy z urządzeniami zewnętrznymi (INP, OUTP, SAVET LOADT, VERIFY).

FP-50 korzysta przy operacjach zmiennoprzecinkowych z kalkulatora Basica ZX Spectrum. Ma więc także zdefiniowane wszystkie funkcje, które zdefiniowano w Basicu. Może on także wykonywać obliczenia na liczbach całkowitych szybciej i zajmując mniej pamięci.

Żadna z prezentowanych wersji nie posiada dołączonego asemblera, nie obsługuje też microdrive'ów.

Forth jest językiem, który można polecić osobom umiejącym dobrze programować w innych językach. Jest trudny i nietypowy, nie można się go nauczyć w kilka dni. Według czasopisma BYTE, wiele uczelni zamyka przed nim drzwi uważając, że jest nieelegancki i operuje (jak na język wysokiego poziomu) na prymitywnych strukturach danych.

* * Pascal 8.7

Cechy języka 8.7.1

Język Pascal reprezentuje odmienny względem Basica sposób programowania. Inna jest struktura programu, inny jest sposób jego wykonania.

Program w Basicu składa się z instrukcji umieszczonych w kolejnych, numerowanych liniach. Zmiana kolejności wykonywania programu następuje poprzez skok do linii o określonym numerze, a więc pominięcie pewnego ciągu instrukcji, gdy nie spełniony jest pewien warunek. Powoduje to nieczytelność programu, wymusza na programiście obowiązek pamiętania, do której linii nastąpił skok i dlaczego. Instrukcje skoku są rozkazami bardzo często pojawiającymi się w programie napisanym w Basicu. Uważa się, że Basic przez swój brak struktury uczy złych nawyków programowania.

Zapis większości algorytmów znacznie łatwiej jest zawrzeć w postaci pewnych struktur, bloków odpowiedzialnych za wykonanie określonych jego części. Zrealizowanie algorytmu polega wtedy na systematycznym przechodzeniu przez kolejne struktury i realizowaniu ich zawartości. Możliwa jest komunikacja pomiędzy tymi blokami, struktury te mogą generować wyniki, które będą danymi dla innych, itd.

Takie podejście do programowania nazywa się programowaniem strukturalnym. Język Pascal posiada mechanizmy programowania strukturalnego. Zdaniem autorytetów stanowi on najlepszy język do nauki programowania.

Program napisany w Pascalu ma bardzo przejrzystą formę — części odpowiadające za realizację fragmentów algorytmu mają wyraźnie wyznaczone granice, instrukcje iteracyjne trzech typów ułatwiają organizację pętli. Wszystkie działania iteracyjne mogą zostać opisane w sposób rekurencyjny, bowiem procedury mogą wywoływać same siebie.

Język programowania Pascal powstał w 1973 r. Jego autorem jest Niklaus Wirth, jeden z najwybitniejszych informatyków. Pascal jest obecnie powszechnie używanym językiem i to zarówno na dużych, jak i na małych, domowych komputerach. Podkreślić należy łatwość implementacji Pascala, jego wszechstronność, możliwość zapisania w tym języku w prosty sposób szerokiej klasy algorytmów.

Pisanie programów w Pascalu jest niewątpliwie trudniejsze niż w Basicu, ze względu na rozbudowane struktury i ścisłą, trudną dla początkujących składnię i semantykę. Nie można wprowadzać poprawek do programu i natychmiast (jak w Basicu) sprawdzić ich działania. Pascal ma jednak dość bogatą diagnostykę błędów, także w fazie wykonania.

Dzięki swoim zaletom Pascal stał się językiem opisu algorytmów nienumerycznego przetwarzania danych. Powstało wiele jego wersji i uzupełnień w zależności od zastosowań. Stosowany jest on do tworzenia i analizowania struktur danych, opisu modeli zjawisk, budowy list, edukacji, opisu i budowy systemów operacyjnych (Concurrent Pascal).

Sukcesorem Pascala jest nowy język Wirtha — Modula 2.

Translator języka Pascal dla ZX Spectrum jest chyba najlepszą z alternatyw interpretera języka Basic, dzięki szybkości wykonywania programu i czytelności zapisu.

Język Pascal jest w Polsce popularny. Na jego temat ukazało się kilka książek oraz wiele artykułów w prasie fachowej. W związku z tym opis składni i struktur języka nie będzie tu prezentowany, jedynie w opisie implementacji zostaną podane odstępstwa i różnice względem standardu języka. Po informacje na temat sposobu programowania w Pascalu pozwalamy sobie odesłać Czytelnika do literatury [10, 30].

Opis implementacji 8.7.2

Translator języka Pascal dla ZX Spectrum jest sprzedawany przez firmę Hisoft. Powstał w 1983 r., jest stale ulepszany, obecnie najpopularniejsza jest wersja HP4T1.4. W skład sprzedawanego zestawu wchodzi także edytor i biblioteka procedur związanych z wykonywaniem programu. Ten sam translator sprzedawany jest także w wersjach na inne mikrokomputery wyposażone w procesor Z80. Do programów dołączona jest instrukcja opisująca szczegółowo cechy translatora.

Organizacja pamięci

Ponieważ po załadowaniu translatora do pamięci wstępny dialog dotyczy dostępnych dla niego adresów, na początek zostanie prześledzony sposób przydziału pamięci i rozlokowanie w niej poszczególnych części translatora.

Cały program składa się z 3 części zajmujących odpowiednio translator — 12, edytor — 2 i biblioteka procedur — 4 kilobajty. Postać źródłowa programu użytkownika opracowywana za pomocą edytora jest przechowywana w buforze edytora, natomiast program po translacji jest wpisywany do obszaru pamięci, którego adres jest wypisywany na ekranie w trakcie tłumaczenia.

Zmienne i dane są przechowywane na stosie roboczym programu, którego adres początku zależy od wartości podanych w odpowiedzi na pytania: Top of RAM? i Top of RAM for T? Pierwsza z nich określa początek stosu kompilatora, druga — stosu obsługi wykonania programu. Gdy ich deklarację pominie się, to zostanie przyjęty adres wskazujący początek obszaru UDG. Obniżając początek stosów można przez obszar ponad nimi przekazywać dane i parametry pomiędzy różnymi programami, nie tylko w Pascalu. Użytkownik może także zażądać przydziału określonej liczby bajtów na tabelę symboli, konieczną do pracy translatora. Jeżeli odpowiedź na pytanie Table size? zostanie pominięta, zajmie ona 1/16 dostępnej pamięci RAM. Należy pamiętać o tym, że deklaracja zbyt dużej tabeli (gdy jej górny adres przekroczy 32768) spowoduje powtórzenie serii pytań, natomiast zbyt małej — błąd kompilacji i konieczność ponownego wczytania translatora, aby zmienić jej wielkość.

W Pascalu istnieją dwa sposoby przydziału pamięci na zmienne — statyczny i dynamiczny. Zmienne deklarowane statycznie w bloku głównym są umieszczane na początku stosu programu, natomiast zmienne lokalne są wskazywane przez rejestr IX procesora (IX-4). Adresuje on także parametry przekazywane pomiędzy blokami. Wskaźniki związane ze zmiennymi tworzonymi dynamicznie zawierają ich fizyczne adresy.

Liczby stałoprzecinkowe w Pascalu zapisywane są na dwu bajtach w kodzie U2 (zakres: −32768 +32767), a liczby zmiennoprzecinkowe na czterech bajtach (inaczej niż w Basicu!) (zakres 5.9E-39, 3.4E+38). Mają one tu tylko 7 cyfr znaczących (stąd 1,00001−1,0= 0). Z kolei znaki alfanumeryczne zajmują po jednym bajcie, a wskaźniki (adresy) — po dwa. Jedynie zbiory przechowywane są jako ciąg bitów (zajmują 8 bajtów). Wartości liczbowe mogą być przekazywane w postaci liczb heksadecymalnych.

Odstępstwa od standardu języka

Są trzy podstawowe ograniczenia:

— nie można używać rekordów z wariantami,
— procedury i funkcje nie mogą być parametrami innych procedur,
— ze względu na brak szybkiej pamięci zewnętrznej nie ma plików (instrukcje READ i WRITE odnoszą się odpowiednio do klawiatury i ekranu). Instrukcje GET, PUT nie są zrealizowane.

Oprócz tego wskaźniki nie mogą bezpośrednio wskazywać na inne wskaźniki. Nie są to istotne ograniczenia. Wprowadzono natomiast wiele rozszerzeń.

Odstępstwa od standardu są następujące:

— nazwy mogą być dowolnej długości, przy czym pierwszych 10 znaków jest znaczących. Dopuszcza się litery duże i małe (słowa kluczowe pisze się dużymi literami),
— definiowana stała może być także dowolnym znakiem, np. CONST cr = CHR(13) — znak o kodzie 13,
— zbiór znaków obejmuje pełen zestaw (kody 0—255) używany przez ZX Spectrum, przy czym znaki o kodach mniejszych od 30 są traktowane jako sterujące (13 — EOLN, 0 — znak NULL, 12 — PAGE (nowa strona), 15 — przejście na drukarkę, bądź zgodnie ze znakami sterującymi Basica np. 1 — INK). Pascal rozpoznaje znaki o kodach w zakresie 30÷127, przy czym przez funkcję CHR (kod) można także drukować znaki zdefiniowane w Basicu jako UDG (znaki A÷U w trybie G).
— wewnątrz tablicy znakowej lub łańcucha znaków nie może wystąpić znak o kodzie 13 (EOLN),
— typy wyliczane mogą mieć do 256 elementów, podobnie jak podstawa zbioru (SET),
— zmienne sterujące pętlą FOR mogą być tylko zmiennymi prostymi,
— instrukcja skoku GOTO nie może przerwać pętli FOR.

Dla procedur i funkcji odstępstwa są następujące:
— procedura WRITE może być formatowna zgodnie ze standardem języka,
— wprowadzono dodatkową funkcję INCH, która podaje kod znaku odpowiadającego ostatnio wciśniętemu klawiszowi (lub 0 jeśli nie wciśnięto żadnego klawisza),
— dołączona biblioteka zawiera podprogramy funkcji matematycznych i konwersji liczb,
— zapamiętanie stanu pamięci stosu przy tworzeniu zmiennej dynamicznej instrukcją NEW (p) dokonuje się przez MARK (p); zwalnianie pamięci — RELEASE (p),
— dodano procedurę INLINE, pozwalającą na wprowadzanie instrukcji maszynowych do wnętrza programu — INLINE (*c*1, *c*2...); *ci* — kolejne bajty (mogą być w zapisie heksadecymalnym). Można także wywoływać zapisany w pamięci program maszynowy przez USER (*adres programu*),
— umożliwiono, tak jak w Basicu, wpisywanie instrukcją POKE bajtów w podane miejsca pamięci, a także ich odczytywanie przez funkcję PEEK,
— wprowadzono funkcje określające adres i długość zmiennej (ADDR i SIZE),
— wyprowadzanie lub wprowadzanie danych z taśmy odbywa się za pomocą procedur TOUT i TIN. Można także współpracować z dowolnym portem we/wy — OUTP i INP.

Uwaga *Przed pierwszym wczytaniem danej, za pomocą instrukcji READ, wskaźnik czytania ustawiony jest na znak końca linii (EOLN). Próba wczytania liczby lub łańcucha znaków spowoduje błąd. Dlatego pierwszą instrukcją czytania powinno być READLN.*

Do wersji HP4T1.4 na kasecie jest dołączony także zestaw podprogramów TURTLE umożliwiających tworzenie grafiki przez sterowanie kursorem. W podręczniku Hisofta (podobnie jak w LOGO) zamieszczono podprogramy pozwalające poprzez procedury systemowe Basica generować dźwięk i rysować.

Edycja i uruchamianie programu

Edytor dołączony do translatora Pascala jest prawie identyczny z oferowanymi wraz z innymi translatorami firmy Hisoft. Ma on dodatkową instrukcję zapamiętywania programu na taśmie (W) w formacie odpowiednim dla późniejszego wczytania go nie przez edytor, a przez kompilator. Przy korzystaniu z edytora należy pamiętać o jednym: wszystkie instrukcje i operatory wprowadza się przez wciśnięcie oddzielnych klawiszy dla każdego znaku (inaczej niż w ZX Basicu), choć faktycznie w pamięci program jest zapamiętany w postaci skompresowanej — każdemu słowu kluczowemu odpowiada jeden bajt. Dotyczy to także operatorów: ⟨⟩, ⟨ =, itd, które wymagają podania dwu znaków. Użycie pojedynczego symbolu ⟨⟩ zamiast dwu ⟨ i ⟩ może spowodować zawieszenie się systemu*).

*) Błąd ten poprawiono w wersji HP4TM1.5.

Opracowany program można przetłumaczyć przez wykonanie dyrektywy C lub T. Ta druga powoduje usunięcie translatora z pamięci, a na jego miejsce wpisanie kodu wynikowego programu (może on być dłuższy niż przy C). Kod ten można zapamiętać na taśmie magnetofonowej. Po przejściu do Basica (B) przetłumaczony program użytkownika można uruchomić instrukcją RANDOMIZE USR 24608.

Uwaga *Przy translacji długich programów, nie mieszczących się w pamięci, przy użyciu dyrektywy C, po wykorzystaniu T czasem mogą pojawić się błędy wykonania.*

Kompilator w trakcie pracy uwzględnia tzw. opcje, to jest dwustanowe znaczniki, opisujące czy w trakcie wykonywania programu sprawdzać przekroczenie adresów pamięci, stosu, nadmiary w operacjach arytmetycznych, itd. Ich wyłączenie powoduje przyśpieszenie pracy programu. Istnieje także opcja F, pozwalająca na dołączanie do programu bloków zapisanych na taśmie dyrektywą W (nie zawsze działa prawidłowo). Kontrola przekroczenia pamięci nie zawsze jest w tym translatorze skuteczna.

Od zeszłego roku sprzedawana jest wersja HP4T1.4M współpracująca z Interface 1 i microdrive'ami.

Translator Pascala firmy Hisoft daje użytkownikowi możliwość nie tylko nauki programowania w tym, tak na świecie popularnym języku. Jest on także dobrym narzędziem pracy dla wykorzystujących Spectrum nie tylko do zabawy.

Pascal na Spectrum jest bardzo szybki. Daje najszybszy kod wynikowy z oferowanych na ten mikrokomputer kompilatorów. Program testowy, który 10 razy wyszukuje liczby pierwsze z 8190 liczb działał na nim zaledwie 34 s. Na świecie rozpowszechnione są dwa typy translatorów Pascala. Pierwszy, prostszy, tłumaczy program na pewien język (kod) pośredni, którego instrukcje są w czasie wykonania interpretowane. Drugi tłumaczy program bezpośrednio do kodu maszynowego, dzięki czemu programy przetłumaczane przez kompilator działają szybciej, ale sam program tłumaczący ma bardziej złożoną budowę.

Należy sądzić, że HP4T jest właśnie takim translatorem. Same tłumaczenie programu odbywa się w nim szybko i tylko w jednym przebiegu.

A oto przykład jego działania:

Adres	Instrukcja asemblera	Instrukcja Pascala
	call #642D ld hl, #5688 jp #A906	PROGRAM A; VAR i: INTEGER
A90C	ld (73C5), hl	BEGIN {początek bloku — ustawienie wskaźników stosu programu, rezerwacja miejsca na zmianę}

```
A915    ld ix, #634E
        ld sp,ix
        ld hl,1              FOR i:= 1 TO 100 DO
        ld (#C34E), hl       {początek pętli}
        jp #A925
A91E    ld hl, (#C34E)
        inc hl
A925    ld de,100
        ex de,hl
        or a                 {sprawdzanie warunku
                             końcowego}
        sbc hl,de
        jp m, #A943
        call #656F
        jp #A938
A935    „ala"
A938    ld b,3               WRITE („ala")
        call #64E8
        jp, #A91E            {koniec pętli}
A943    jp, #6041            END.
A946    —                    {koniec programu}
```

Translator pozostawia dużo miejsca na program użytkownika. Szkoda jednak, że nie pozwala na łączenie ze sobą przetłumaczonych bloków. Ma on pewne niedociągnięcia, ale w sumie jest godny polecenia. Pozwala nauczyć się pisania programów w strukturalnym języku wysokiego poziomu, nadaje się także do profesjonalnych zastosowań, np.: sterowania urządzeniami, obliczeń i in. Wadą Pascala na ZX Spectrum jest mało czytelna diagnostyka błędów w fazie wykonania (zamiast numeru linii drukowany jest adres instrukcji w pamięci).

* * Język C 8.8

Cechy języka 8.8.1

Jednym z ciekawszych kompilatorów języków wysokiego poziomu napisanych dla ZX Spectrum jest translator języka C. Został on zrealizowany w 1984 r. przez firmę Hisoft.

Jest to język wysokiego poziomu (jak Pascal, Basic, Fortran), tłumaczony do kodu maszynowego. Zawiera on konstrukcje charakterystyczne dla tych języków (takie jak: for, while, if, else, do), możliwości strukturalizacji programu, a także tworzenia skomplikowanych struktur danych. Równocześnie jednak ma wbudowane mechanizmy charakterystyczne dla asemblera, np.: określanie z poziomu programu w języku C sposobu wykorzystania poszczególnych rejestrów

procesora, różne tryby adresowania, operacje odpowiadające instrukcjom maszynowym (np.: inkrementacja*) zmiennej). To połączenie instrukcji wysokiego i niskiego poziomu pozwala pisać bardzo efektywne programy.

Język C jest szeroko stosowany tam, gdzie trzeba pisać bardzo duże programy, działające szybko i niezawodnie, a więc głównie oprogramowanie systemowe (i to dla dużych komputerów). Obecnie pisze się w C te programy, które dawniej pisano w asemblerze, a to dlatego, że daje się on bardzo efektywnie tłumaczyć na kod maszynowy. Programy napisane w tym języku wykonuje się tylko niewiele wolniej niż stworzone w asemblerze, zajmują też tylko nieco więcej pamięci.

Pamiętać należy, że sprzęt (pamięć i procesor) ze względu na stałą obniżkę jego cen stanowi obecnie coraz mniejszy procent kosztu całego komputera — gros wydatków pochłania jego oprogramowanie. Zatem trzeba je pisać jak najtaniej — szybko, ale i efektywnie. Pisane w języku wysokiego poziomu byłoby dobrym rozwiązaniem, niestety programy takie działają zwykle zbyt wolno. I tu jest właśnie miejsce na język C, stanowiący kompromis między tymi żądaniami, bo dający łatwość pisania typową dla języków wysokiego poziomu połączoną z szybkością wykonywania porównywalną z asemblerem.

Język C został bardzo precyzyjnie zdefiniowany, dzięki czemu implementacje tego języka na różnych komputerach są niemal identyczne. Program napisany w C np. na IBM PC może być bez zmian używany na innym komputerze, w przypadku innych języków jest to niemożliwe.

O jednej z cech, współdziałaniu instrukcji charakterystycznych dla języków wysokiego i niskiego poziomu, już wspomniano. Trzeba stwierdzić, że pisanie programów jest tu znacznie bardziej skomplikowane niż w Basicu, ze względu na takie czynniki jak:
— rozbudowane struktury danych,
— łączenie operatorów w wyrażenia,
— struktury programu,
— brak mechanizmów kontroli operowania na zmiennych (utrudnia to uruchamianie programów).

Składnia języka może sprawiać wiele kłopotów nawet zaawansowanym programistom. Ma on wiele operatorów, niektóre z nich przyjmują w zależności od kontekstu różne znaczenie (np.: „*" — operator mnożenia, adresowania pośredniego; „(*" i „*)" — znak komentarza).

Biblioteka języka C (zbiór procedur zdefiniowanych przez twórców systemu) jest na ogół bardzo bogata i oprócz algorytmów różnych obliczeń zawiera także podprogramy sortowania danych, konwersji liczb, itp.

O przydatności języka C do pisania oprogramowania systemowego świadczy fakt, że najpopularniejszy obecnie system operacyjny dla minikomputerów UNIX został napisany (w znacznej części) właśnie w języku C.

Czytelników zainteresowanych językiem C odsyłamy do literatury [29].

*) Zwiększenie o jeden.

Firma Hisoft sprzedaje na kasecie magnetofonowej nie tylko translator C, lecz także całe otoczenie programowe, konieczne do pisania programów w tym języku, to jest edytor i bibliotekę systemową.

Edytor jest bardzo zbliżony do programów edycyjnych dodawanych przez tę firmę do innych języków (Pascal, Asembler GENS3). Realizuje on podstawowe funkcje takie jak wprowadzanie, kasowanie i przesuwanie linii programu, ich numerację, wyszukiwanie podanych ciągów znaków, itd. Zawiera także tzw. edytor kontekstowy umożliwiający poprawianie znaków w wybranej linii tekstu. W sumie edytor można uznać za wystarczający.

Podstawowym ograniczeniem tej wersji C jest brak implementacji liczb i operacji zmiennoprzecinkowych, co nakłada daleko idące ograniczenia na typy i struktury danych, operacje arytmetyczne i operatory. Nie wprowadzono także liczb typu *long integer* (4-bajtowych liczb całkowitych). Jedyne dopuszczalne typy danych to liczby całkowite 2-bajtowe (*integer*) i 1-bajtowe znaki (*char*). Jest to niewątpliwie poważne ograniczenie zastosowań tego translatora. Arytmetyka jest zatem 16-bitowa (za wyjątkiem generacji przebiegów pseudolosowych, gdzie używa się operacji 32-bitowych)*).

W translatorze dopuszczono możliwość wykonywania instrukcji języka w trybie natychmiastowym, co jest dla realizacji tego języka dość nietypowe. Wykonanie w tym trybie jest równoważne wykonaniu instrukcji Basica podanej bez numeru linii.

Jak już wspomniano, dość bogata jest biblioteka systemowa, która zawiera podprogramy obliczeń, konwersji, przydziału pamięci, współpracy z urządzeniami wejścia-wyjścia. Nie ma, niestety, możliwości dołączania do biblioteki własnych procedur, zrezygnowano ponadto z translacji warunkowej.

Liczne cechy implementacji są związane z organizacją ZX Spectrum. Tak więc dopuszczono możliwość korzystania z 3 źródeł danych: taśmy magnetofonowej, microdrive'u i łącza szeregowego RS 232 (dwa ostatnie przy podłączonym Interface 1). Pliki zewnętrzne są dołączane do systemu w sposób standardowy, to jest przez kanały i strumienie. Do biblioteki języka dodano procedury graficzne i generacji dźwięku odpowiadające analogicznym instrukcjom Basica. Liczne ograniczenia typowych dla C instrukcji wynikają z ograniczonej pamięci i słabego procesora. Stąd np. zmienne zadeklarowane jako rezydujące w rejestrach mikroprocesora są umieszczane nie w nich, a w pamięci, bo rejestrów Z80 jest po prostu za mało.

Aby umożliwić pisanie nieco dłuższych programów wprowadzono opcję translacji programu i wyprowadzania jej produktu, łącznie ze wszystkimi potrzebnymi do jego uruchamiania procedurami, na taśmę. Tak przetłumaczony program można następnie wykonać tak jak napisany w języku wewnętrznym (RANDOMIZE USER 25200).

*) Ostatnio (koniec 1985 r.) mówi się o pojawieniu się zmiennoprzecinkowej wersji C. Niestety, autorzy mogli testować tylko zubożoną wersję.

W pamięci Spectrum oprócz translatora i programu w C można zmieścić króciutki program napisany w Basicu.

Mimo tak wielu ograniczeń nałożonych przez twórców, translator należy ocenić wysoko. Napisanie takiego programu jest wyczynem nie lada, bowiem jest to język przewidziany dla komputerów o kilka klas lepszych od ZX Spectrum, o znacznie większej pamięci i szybszym, doskonalszym procesorze.

W fazie translacji i łączenia programu kłopot może sprawić dołączanie procedur z taśmy magnetofonowej, gdyż w porównaniu z bardzo szybkim tłumaczeniem programu, ładowanie i wyszukiwanie na taśmie podprogramów bibliotecznych trwa o wiele za długo. Rozwiązaniem jest dołączenie microdrive'u lub korzystanie z innych urządzeń podłączonych przez łącze szeregowe.

Przykładem szybkości obliczeń w języku C może być 10-krotne wykonanie algorytmu Eratostensa wyszukiwania wśród 8190 liczb liczb pierwszych. Taki program wykonuje się około 60 s. To jest dobry wynik, ale gorszy niż dla translatora Pascala, co nie jest dobrą reklamą dla C, ale znakomitą dla Pascala (także firmy Hisoft)!

Korzystanie z translatora języka C ułatwia dobrze napisana i obszerna instrukcja, a także stosunkowo nieźła diagnostyka błędów wykonania programu.

Reasumując, jest to dobry translator, szczególnie do nauki programowania w języku C i rozwiązywania problemów przetwarzania danych, obsługi urządzeń zewnętrznych oraz pisania prostych programów systemowych.

* * Prolog 8.9

Cechy języka 8.9.1

Wszystkie prezentowane do tej pory języki różniły się między sobą sposobem podejścia do przedstawionego w nim zagadnienia, a szczególnie sposobem powiązania teoretycznych rozważań z maszyną, komputerem, który ma dany problem rozwiązać. W tym sposobie przedstawienia algorytmu była jednak pewna cecha wspólna, sposób rozwiązania zapisany był w ciągu procedur koniecznych do jego rozwikłania. W Prologu podaje się najpierw jaki rodzaj rozwiązania jest poszukiwany, a dopiero potem rozwiązuje się go dla konkretnego przypadku, np. aby obliczyć 7^2 najpierw podaje się definicje funkcji kwadrat, a dopiero potem odpowiada się na konkretne pytanie, ile wyniesie wynik dla 7. Taki język nazywa się s p e c y f i k a c y j n y m (ang. *declarative*). Wystarczy, aby przedstawiony problem został w nim opisany za pomocą twierdzeń i reguł wnioskowania w sposób ścisły, a zostanie on automatycznie przez komputer rozwiązany, bez udziału człowieka.

Jest to język interpretowany bardzo wysokiego poziomu, przeznaczony do zapisywania wiedzy na pewien temat i jej użycia, w postaci obiektów (zbioru

własności) i relacji — powiązań pomiędzy obiektami reprezentujących ich wzajemne oddziaływania. Na przykład, wiedzę o sposobach dekoracji filiżanek porcelanowych można przedstawić jako opis cech filiżanek pochodzących z różnych wytwórni, a relację między nimi opisywać będą jaka wytwórnia, w jakim czasie, wzorowała się na innej.

Język Prolog jest całkowicie niezależny od sprzętu. Nie występują w nim takie pojęcia jak rejestry, tablice czy instrukcje. Zamiast tego użytkownik posługuje się pojęciami logicznymi — relacjami między zdefiniowanymi obiektami.

Prolog powstał w 1973 r. na Uniwersytecie Marsylskim i początkowo pomyślany był jako język służący do dowodzenia pewnej klasy twierdzeń matematycznych. Jego idea jest oparta na zdaniu brytyjskiego matematyka Roberta Kowalskiego, że logika jest sposobem przyswajania wiedzy przez człowieka, a co za tym idzie w języku logiki najłatwiej jest sprecyfikować problemy i programować ich rozwiązywanie.

Pierwsze interpretery Prologu powstałych dla dużych komputerów, w ciągu 10 lat przeniesiono go na mini- i mikrokomputery, tworząc wersję micro-Prolog*) (McCabe).

Od kilku lat Prolog jest modny. Przyczyną tego są badania nad tak zwaną „sztuczną inteligencją". Realizacja tego projektu ma umożliwić powstanie, w ciągu najbliższej dekady, komputera piątej generacji, który będzie potrafił uczyć się, wnioskować i porozumiewać się z człowiekiem za pomocą pisanego języka naturalnego, bądź mowy.

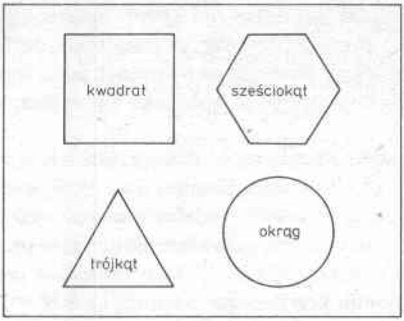
Prolog stosuje się obecnie tam, gdzie potrzebny jest opis wiedzy na dany temat. W zagadnieniach sztucznej inteligencji, do komunikacji z komputerem w języku naturalnym, w bankach danych, do tworzenia kompilatorów, do programowania robotów, systemów doradczych (ang. *expert system*), wspomagania projektowania.

Zainteresowanie Prologiem spowodowało dynamiczny jego rozwój, powstaje coraz więcej wersji tego języka. Na początku lat 80-tych w Imperial College w Londynie prowadzono eksperyment, który polegał na nauce dzieci, w wieku 8—13 lat, programowania w tym języku w uproszczonej notacji SIMPLE. Z drugiej strony profesjonaliści tworzą wersję Prologu umożliwiającą testowanie hipotez, a więc zagadnień nie opisanych do końca. Są także wersje, które, gdy same nie potrafią rozwiązać problemu, proszą o pomoc. Są też takie, które umieją rozwiązywać postawione zagadnienia równolegle.

Ze względu na skomplikowany sposób pracy Prolog działa dosyć wolno. Ale już i na to znaleziono radę — tworzy się specjalne procesory (nie mikroprocesory) służące do jego implementacji.

Jak działa Prolog? Objaśnimy to na przykładzie (według Clark, Mc Cabe [3]); zapis podano w notacji SIMPLE. Na rysunku 36 przedstawiono 4 figury geometryczne. Jak zapisać wiedzę o ich wzajemnym położeniu? Najpierw tworzy się bazę danych. Będzie ona zawierać informację o obiektach i ich relacjach:

*) Nazwa micro-Prolog oddaje tylko rodzaj sprzętu, na który został przeniesiony, bowiem jego możliwości pozostały w skali makro.



Rys. 36. Przykład

— square *above* triangle (kwadrat ponad trójkątem),
— hexagon *above* circle (sześciokąt ponad okręgiem),
— hexagon *right* square (sześciokąt na prawo od kwadratu),
— circle *right* triangle (okrąg na prawo od trójkąta).

Obiekty to: square, hexagon, triangle, circle. Relacje łączące obiekty to: *above* i *right*. W powyższy sposób zdefiniowano wiedzę na temat położenia figur. Tę wiedzę można wykorzystać do zadawania pytań — można poprosić o wydrukowanie spisu tych figur, które są po prawej stronie, można pytać, czy dany element jest nad innymi. Odpowiedź brzmi: tak lub nie, przy czym interpreter uważa relację za spełnioną tylko wtedy, gdy wszystkie jej elementy są spełnione. Jeżeli brak jest elementu spełniającego, to relację uważa się za fałszywą.

W Prologu można także definiować zmienne (zaczynające się od x, y, z). Można np. zapytać *is* (x *above* triangle) — czy coś jest nad trójkątem — *YES*, square *above* triangle (kwadrat ponad trójkątem). Można też zapytać czy ,,coś" jest nad ,,czymś" — *which* (x:x *above* y), tzn. jakie elementy należą do zbioru określonego relacją *above*. Odpowiedź: square and hexagon (kwadrat i sześciokąt). Interpreter Prologu po zadaniu takiego pytania podstawia zamiast zmiennych kolejne elementy bazy danych. Czyni to metodą zstępującą, to znaczy najpierw poszukuje obiektów najbardziej złożonych, potem coraz prostszych. Jeżeli podstawienie jest niespełnione, to jest ono ,,wycierane", a w to miejsce jest podstawiany następny kandydujący obiekt.

Struktury obiektów mogą być bardzo skomplikowane, ale zawsze są powiązane między sobą pewnymi relacjami. Prolog w trakcie rozwiązywania problemu przeszukuje wszelkie możliwe drogi w tych strukturach danych — dlatego trwa to długo. W skomplikowanych programach, aby skrócić czas obliczeń, pewne przejścia przez struktury można zablokować. Programista musi wiedzieć, że dalsze podążanie tą ścieżką prowadzi na manowce.

Program napisany w Prologu ma bardzo zwartą formę i przy zastosowaniu do dużych problemów jego wykonanie może trwać krócej niż programu zapisanego

w języku kompilowanym. Program jest budowany stopniowo, procedury dostępne w danej chwili stanowią jego treść.

Uważa się, że języki programowania zbliżone do Prologu to języki jutra, gdyż opisują wiedzę, nie są uzależnione od sprzętu, pozwalają na matematycznie ścisłe wnioskowanie. Nie nadają się jednak do obliczeń inżynierskich, są przeznaczone do rozwiązywania problemów. Opis języka można znaleźć w pracach [2, 14] oraz w numerach 2, 3 i 4 *Bajtka*.

Opis implementacji 8.9.2

Wersja micro-Prologu na Spectrum została stworzona przez firmę Logic Programming Associates w 1983 r. W porównaniu z Prologiem stosowanym na dużych komputerach różnice dotyczą głównie składni i dopuszczalnych form pytań (według przykładu *is*, *which*). Ogromne możliwości języka mogą być zaskoczeniem, gdyż tak skomplikowany interpreter jest zawarty w tak małej pamięci, pozostawiając około 25 kB na bazę danych użytkownika. Trzeba przy tym zaznaczyć, że oprócz procedur współpracy z pamięcią zewnętrzną (dla ZX Spectrum — taśma magnetofonowa) nie różni się on w zasadzie niczym od starszych wersji programu dla IBM PC, BBC micro, czy komputerów działających w systemie UNIX!

Jest to program sprzedawany w wyjątkowo bogatej szacie. Po pierwsze, oprócz translatora na kasecie firmowej znajduje się zestaw około 10 programów umożliwiających dokładne poznanie możliwości języka, łącznie z programem wprowadzającym uproszczoną notację SIMPLE. Po drugie, oprócz broszury wyjaśniającej sposób ładowania programu do Spectrum i prostego przykładu pokazującego jego możliwości, dołączana jest książka twórców języka opisująca na blisko 300 stronach w sposób szczegółowy metody działania, gramatykę i struktury danych języka („ZX Spectrum micro-Prolog primer").

Ta niezwykła staranność edycji programu jest adekwatna do jego ceny, a wynika to chyba z prób stosowania tej wersji do nauki dzieci. Ale jest on niewątpliwie warty swojej ceny. Dla osób stykających się z problemami matematyki, baz danych, testowania hipotez, szybko stanie się on niezastąpionym narzędziem.

Utworzone struktury danych i programy można zapamiętywać na taśmie magnetofonowej. Programy mogą być wykonywane w sposób interakcyjny, a więc w formie dialogu z komputerem. Jest to ważne przy tworzeniu np. baz danych i umożliwianiu użytkownikowi zadawania pytań o kolejne jej elementy. Można także w Prologu na Spectrum tworzyć programy nadzorujące pracę innych programów, można pisać translatory własnych języków. W ogóle można bardzo dużo, znacznie więcej (w sensie abstrakcji) niż w jakimkolwiek innym języku na ZX Spectrum.

Niestety, nia ma tu miejsca na szczegółowe opisywanie struktur języka. Warto jednak wspomnieć o możliwości definiowania modułów (rodzaj podprogramów),

procedur rekursywnych (wywołujących same siebie), operacji stało- i zmiennoprzecinkowych, tworzenia struktur danych — drzew i list.

* * Asemblery i Disasemblery 8.10

Jeśli program musi działać bardzo szybko i efektywnie, a nie wystarcza programowanie w językach C lub Forth, to wtedy sięga się po asembler. Oczywiście, zapisywanie długich programów w języku wewnętrznym przez bardzo niewygodne PEEK i POKE nie ma sensu. Trzeba użyć programu tłumaczącego mnemoniki instrukcji procesora na jego kody, pozwalającego ponadto na umieszczanie kodu w wybranym miejscu pamięci, jego zmiany i testowanie. Program taki, dla odróżnienia od samego języka (asembler), nazywany będzie Asemblerem.

Wiele firm produkujących oprogramowanie opracowało Asemblery dla ZX Spectrum. Jednak jedynie Hisoft sprzedaje pełny, zintegrowany zestaw narzędzi koniecznych do pisania programów w języku wewnętrznym, tzn. Asembler z edytorem i Monitor-disasembler (istnieje jeszcze, mało w Polsce popularny, zestaw firmy Picturescue). Dla wyjaśnienia: monitorem nazywa się program umożliwiający testowanie i uruchamianie innych programów, wyświetlanie zawartości pamięci, rejestrów procesora. Monitor umożliwiający dodatkowo dekodowanie programów asemblerowych (z kodu wewnętrznego na instrukcje asemblera) nazywany jest disasemblerem. Programy tego typu Anglicy nazywają także debugger'ami, uważają oni bowiem oczyszczanie programów z błędów za czynność zbliżoną do (dosłownie)... odpluskwiania.

W niniejszym rozdziale zostanie przedstawiony zestaw DEVPAC 3 firmy Hisoft (Asembler GENS3 i Monitor MONS3)*).

Asembler

Język asemblera Z80 został opisany w rozdziale 4.3. Dla przypomnienia, linia programu napisanego w nim ma swój charakterystyczny format: pole etykiety, mnemonik instrukcji, argumenty, ewentualny komentarz. Komentarz zaczyna się od znaku ,,;'' w 1 lub 32 kolumnie. Każdy z programów tłumaczących taki język jest wyposażony w integralny program edycji jego postaci źródłowej (tekstu).

Edytor GENS3 jest zbliżony do programów tego typu dołączonych do innych translatorów firmy Hisoft. Ma on instrukcje umożliwiające:
— wprowadzanie i numerowanie linii (I)**),
— usuwanie linii (D),

*) O ile MONS3 nie ma w zasadzie konkurenta, o tyle istnieją Asemblery lepsze niż GENS.

**) Ze względu na niewielką liczbę instrukcji autorzy pozwolili sobie na podanie przykładów ich nazw.

— przepisywanie (M),
— listowanie tekstu na ekran lub drukarkę (L,W).

Wprowadzane linie można automatycznie formatować. Naciskając klawisz można przesuwać się do kolejnych pól w linii. Edytor ma instrukcje pozwalające zapisać treść programu na taśmie magnetofonowej (P), także w formacie wymaganym przez Asembler (T). Treść programu można wczytać z taśmy (G). Edytor jest dodatkowo wyposażony w tzw. edytor kontekstowy, umożliwiający korektę fragmentu linii. Ciąg znaków występujący w programie można wyszukać (F) i poprawić w prosty sposób, kopiując linię do edytora kontekstowego i ustawiając wskaźnik (kursor) w odpowiednim miejscu. Znaki w linii można wymieniać (C), usuwać (K) i dopisywać (I).

Po załadowaniu GENS3 do pamięci na ekranie pojawia się pytanie: Buffer size? Odpowiedź (0÷9) reguluje wielkość bufora dla wprowadzonego tekstu (pominięcie powoduje przyjęcie bufora o wielkości 256 bajtów). Z edytora można powrócić do systemu Basica (B).

Cały program GENS3 jest napisany w taki sposób, że może zostać załadowany w dowolne miejsce pamięci. Zajmuje on około 8 kB, ale musi mieć miejsce na tekst programu i tabelę symboli (konieczną przy asemblacji).

Tłumaczenie przygotowanego w buforze edytora programu dokonuje się po podaniu polecenia A i przydzieleniu przez użytkownika pamięci (ilość podana w bajtach) na tabelę symboli (asembler potrafi oszacować jej długość). Użytkownik może także określić opcje translacji, tj. czy program ma być podczas pracy Asemblera listowany, czy wydrukować tabelę symboli, gdzie umieszczać kod wynikowy, itd.

Asembler tłumaczy program użytkowania przeglądając go dwukrotnie. Za pierwszym razem wyszukuje wszystkie etykiety, wyrażenia i zapisuje je w tabeli symboli. Podczas drugiego przebiegu, nadaje im wartości i zapamiętuje kod programu w wyznaczonym na niego miejscu,

Oprócz wszystkich instrukcji Z80 w programie użytkowania mogą wystąpić tzw. pseudoinstrukcje. Mają one taki sam format jak instrukcje, lecz nie są zamieniane na instrukcje maszynowe. Mają one charakter pomocniczy — pozwalają umieszczać program w wybranym miejscu pamięci, nadawać wartość etykietom, deklarować zmienne i bufory. Są to:

— ORG *wyrażenie* ustawienie początkowego adresu wprowadzania, np. ORG 60000, oznacza że program będzie wprowadzony do pamięci od adresu 60000,
— *etykieta* EQU *wyrażenie* — nadanie wartości etykiecie, np. CR EQU 13 nadaje stałej CR wartość 13,
— *etykieta* DEFB *wyrażenie* rezerwacja 1 bajtu pamięci i przypisanie mu wartości, np. A1 DEFB 0; Podobnie DEFW rezerwuje 2 bajty, a DEFS *k* — przydziela *k* — bajtów pamięci, rezerwując miejsce np. na dane.

Argumentami instrukcji mogą być stałe lub wyrażenia. Stałe mogą mieć zapis dziesiętny (np. 125), heksadecymalny (#1F), binarny (00011001) lub znakowy ("a"). Wyrażenia to w tym przypadku stałe połączone ze sobą operatorem arytmetycznym lub logicznym.

Translator ma także tzw. dyrektywy, które sterują przebiegiem translacji. Umieszczane są one w programie źródłowym. I tak np. F "*nazwa*" — pozwala dołączać programy źródłowe zapisane na taśmie, S — zatrzymywać wydruk programu do czasu naciśnięcia dowolnego klawisza, itd. W Asemblerze wprowadzono także tzw. translację warunkową. Jeśli pewne wyrażenie jest spełnione, to translowany jest pierwszy ciąg instrukcji, jeśli nie — drugi (IF *wyrażenie instrukcja 1* ELSE *instrukcja 2* END).

Asembler GENS3 ma jednak pewne wady, do których należy zaliczyć*):

— brak instrukcji umożliwiających zapamiętanie przetłumaczonego programu na taśmie,
— konieczność uprzedniego włączenia magnetofonu przed wykonaniem instrukcji P (program jest od razu transmitowany),
— brak instrukcji weryfikacji zapisu na taśmie,
— przepisywanie tylko pojedynczych instrukcji (w edytorze),
— nie najlepszy edytor.

GENS3 jest jednak niezawodny, a ponieważ współpracuje doskonale z disasemblerem MONS3, można go polecić ewentualnym użytkownikom.

Od połowy 1983 r. sprzedawana jest wersja GENS3M umożliwiająca współpracę z microdrive'ami przez Interface 1 i mająca tzw. makrodefinicje (rodzaj procedur).

Monitor — debugger — disasembler

Jak już wspomniano, program tego typu ma za zadanie ułatwić użytkownikowi uruchomienie programu. W jaki sposób? Przede wszystkim dzięki możliwości krokowego, częściowego uruchamiania programu. Odbywa się to za pośrednictwem tak zwanych pułapek. Pod pojęciem tym rozumie się miejsce w testowanym programie, w którym ma nastąpić zatrzymanie jego wykonywania. Można wówczas sprawdzić stan zmiennych, rejestrów procesora lub uruchomić program powtórnie. Podczas pracy programu monitora MONS3 na ekranie stale jest prezentowana zawartość rejestrów procesora (można ją zmieniać), a także po 8 bajtów pamięci adresowanych zawartością kolejnych par rejestrów. Można także przeglądać pamięć ZX Spectrum, traktując jej kolejne bajty jako instrukcje asemblera, bądź jako znaki alfanumeryczne.

Jedną z wygodniejszych instrukcji monitora jest N — służy do wyszukiwania wybranego ciągu danych. Zlecenie T umożliwia współpracę z asemblerem GENS3 — dokonuje disasemblacji do postaci wymaganej przez ten program. Nawet adresom pętli zostają wtedy przypisane etykiety.

MONS3 został wyposażony w zlecenia pozwalające przepisywać bloki pamięci i wypełniać je zadaną zawartością. Kolejno wprowadzone bajty muszą mieć

*) Pod wieloma względami przewyższa go EDIT/ASS firmy OCP, mający tzw. edytor ekranowy (ang. *full-screen editor*). Poprawianie znaku polega tu na ustawieniu kursora na żądany znak w wylistowanym programie i wpisaniu na jego miejsce żądanego tekstu.

wartości zadane w zapisie heksadecymalnym. Instrukcja H pozwala na konwersję liczby dziesiętnej na szesnastkową, podawane adresy mogą być na żądanie wyświetlane w zapisie dziesiętnym.

Cały ten, tak użyteczny program zajmuje około 5 kB, ale ze względu na konieczność rezerwacji pamięci na tabelę relokacji całość monitora trzeba szacować na około 6 kB. Podobnie jak GENS3, disasembler jest relokowalny, tj. można go wpisać w dowolny obszar pamięci RAM. Ma on własny stos i dzięki temu jego działanie jest niezależne od reszty systemu.

Dla porównania, program-monitor MCTT sprzedawany razem z Asemblerem EDIT/ASS nie ma disasemblera (pamięć można obejrzeć tylko w postaci kodów heksadecymalnych) i umożliwia tylko zestawianie pułapek w zadanym miejscu pamięci i modyfikowanie zawartości pamięci.

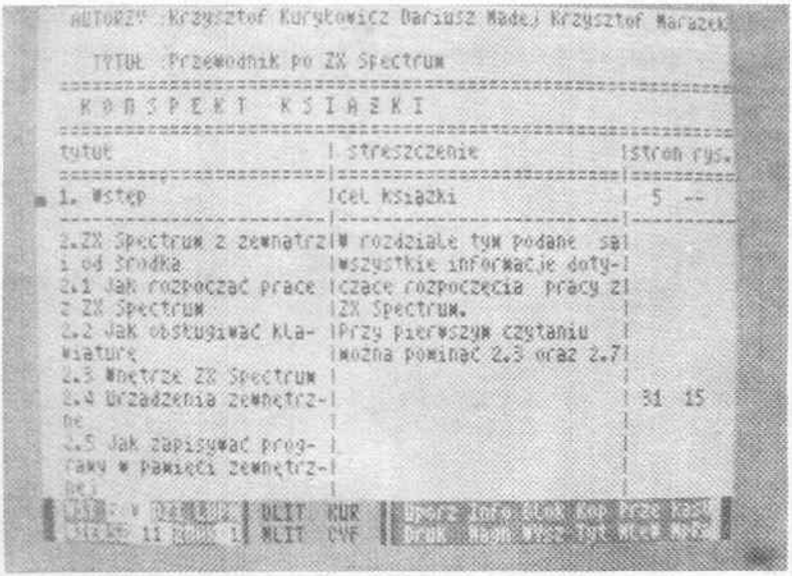
Uwaga *Podczas pracy monitora MONS3 zablokowane jest przyjmowanie przez system przerwań.*

* Programy do pracy, nauki i zabawy 9

* Edytory tekstów 9.1

Edytor tekstów (ang. *word processor*) jest to program, który czyni z komputera inteligentną maszynę do pisania. Jest on jednym z podstawowych programów użytkowych każdego mikrokomputera. Wiele firm zajmujących się sprzedażą mikrokomputerów dołącza ten program jako standardowe wyposażenie swoich zestawów. Przykładów takich komputerów nie trzeba szukać daleko. Należą do nich między innymi zestawy: Sinclair QL lub też Amstrad CPC 464.

W sprzedaży znajduje się kilkanaście programów tego typu dla ZX Spectrum.



Podstawowe z nich to: *Tasword Two* firmy TASMAN oraz *Wordprocessor* firmy MICROL. Dla polskich użytkowników podstawową wadą wyżej wymienionych programów był brak liter polskiego alfabetu. Dlatego też opierając się na programie *Tasword Two* firma POLBRIT opracowała program o nazwie *Procesor/Edytor tekstów 0.1 i 0.2*. W programach tych polskie małe i duże litery dostępne są bezpośrednio z klawiatury*).

Do czego służą edytory tekstów? Podstawowym zadaniem takiego programu jest redagowanie wprowadzonych danych tekstowych oraz ich wydruk. W przypadku programów z rodziny *Tasword* mikrokomputer może zapamiętać do 320 wierszy po 64 znaki w każdym. Średnio na każdej stronie maszynopisu znajduje się 30 wierszy, więc w pamięci mieści się 10 i 2/3 strony.

W czasie pracy z programem edytora ekran telewizora jest ,,oknem" pokazującym 22 linie tekstu po 64 znaki**). Dolne dwie linie ekranu — okno systemowe — przeznaczone są na ciągłe wyświetlanie informacji o wybranych opcjach programu. Ze względu na ograniczoną ilość miejsca w oknie systemowym nie wszystkie dostępne opcje są tam opisane. Są one opisane na tzw. stronie *Help* w programie *Tasword* lub na stronie informacyjnej w programie *Procesor/edytor tekstów*. Dostęp do tych informacji możliwy jest niemal w każdym momencie pracy z programem. Poza tym w czasie korzystania ze strony informacyjnej *Help* nie są niszczone wprowadzone do pamięci komputera dane tekstowe. Odpowiednio zredagowany tekst może zostać wydrukowany na drukarce. Omawiane edytory tekstów mogą pracować z drukarkami ZX Printer (nieudana) oraz z Seikosha GP 50 A, bez dodatkowych programów obsługi drukarek. Inne drukarki jak: D100, DZM 180 (produkcji Mera Błonie), Seikosha GP 500 A lub GP 100 A, czy też SHINWA CTA 80 muszą być podłączone do ZX Spectrum za pomocą odpowiedniego układu dopasowującego. W programie *Procesor/edytor tekstów* wbudowane są na stałe podprogramy obsługi wymienionych drukarek oraz układów dopasowujących, takich jak PB3 (o standardzie Centronics i Logabax firmy Polbrit), ZX Interface 1, RS 232 i Kempston E. Poza tym istnieje możliwość wczytania do programu edytora (na stałe lub jednorazowo) procedury obsługi układu sprzęgającego użytkownika.

Aby współpraca z jednym ze wspomnianych edytorów tekstów odbywała się bez kłopotów, należy przestrzegać dwóch podstawowych reguł:

- Po każdym znaku interpunkcyjnym należy wprowadzić dodatkowo minimum jeden odstęp.
- Zaczynając nowy akapit należy go poprzedzić odpowiednią liczbą odstępów lub nawet linią pustą.

Przed rozpoczęciem wprowadzania tekstu do pamięci mikrokomputera trzeba określić liczbę znaków w wierszu ustalając lewy i prawy margines. Zwykle w jednym wierszu umieszcza się 60 znaków.

*) Nie są to jedyne na naszym rynku programy z polskimi literami wzorowane na *Tasword*, ale są one najbardziej znane.

**) Znaki zapisywane są w polu 8×4 punkty ekranu, dlatego są mniej czytelne.

W kulturze europejskiej przyjęło się, że tekst wygląda elegancko, jeżeli jest opisany z wyrównaniem do prawego marginesu (w przeciwieństwie do kultury arabskiej) oraz jest w nim jak najmniejsza liczba wyrazów przeniesionych z jednego do drugiego wiersza. Program edytora tekstów zapobiega dzieleniu wyrazów między wiersze przy prawym marginesie. Znaczy to, że każdy wyraz nie mieszczący się w danym wierszu jest automatycznie przesuwany o wiersz niżej. Pozostałe w górnym wierszu wyrazy przesuwane są do prawego marginesu przez odpowiednie zwiększenie odstępów między nimi. Może się zdarzyć, że użytkownik nie życzy sobie przeniesienia długiego wyrazu do następnego wiersza, np.: w danym wierszu nie mieści się tylko jedna litera tego wyrazu. Problem ten można rozwiązać dwoma sposobami. Wyłącza się tryb automatycznego dzielenia wyrazów między wiersze oraz wykonuje się równanie do prawego marginesu w obrębie jednego wiersza. Można także po przejściu do nowej linii wprowadzić znak odstępu, który zapobiega dzieleniu wyrazów między wiersze, a po nim dopisać brakujące litery.

W czasie pracy z edytorem tekstów zdarza się przeoczenie informacji, która powinna być umieszczona wewnątrz wprowadzonego do pamięci mikrokomputera tekstu. Uzupełnienie brakujących informacji oraz umieszczenie ich w odpowiednim miejscu wcale nie jest trudne. Informacje te dopisuje się w tzw. trybie dopisywania, a edytor automatycznie rezerwuje miejsce na wstawkę, lub pisze się je na końcu tekstu i oznacza odpowiednimi znacznikami. Następnie przesuwa się kursor ustalający aktualną pozycję wydruku w wybrane miejsce programu. Korzystając z odpowiedniej opcji programu umieszcza się brakujące informacje (wprowadzone na końcu) w miejsce wskazywane przez kursor. Przesunięcie takie może odbyć się wraz ze skasowaniem tekstu źródłowego lub bez jego skasowania. Tę opcję programu można zastosować do zmiany kolejności pisanych zdań lub całych akapitów.

We wspomnianych tu programach opcja ta nie działa najlepiej, nieraz wprowadza ona trochę bałaganu w tekście. Uporządkowanie tekstu wymaga wtedy skasowania lub dopisania kilku słów, zgubionych znaków czy wyrazów. Powstały w ten sposób chaos usuwa się wykorzystując opcję porządkowania akapitu.

Po zakończeniu wprowadzania informacji do pamięci mikrokomputera tekst należy sprawdzić — czytając go. Na ekranie wyświetlane są tylko 22 wiersze. Po przeczytaniu zawartości ekranu można go przesuwać linia po linii lub strona (ekran) po stronie, a znalezione usterki poprawiać na bieżąco. Poza tym od edytora można zażądać znajdowania w tekście dowolnych wyrazów. Wówczas program sam przeszukuje wprowadzone do pamięci informację, a zatrzymuje się tylko wtedy, gdy znajdzie określony przez użytkownika ciąg znaków. Dzięki temu w prosty sposób można sprawdzić, czy dany wyraz nie jest nadużywany.

Każdy wpisany do pamięci mikrokomputera tekst można zapisać w pamięci zewnętrznej, czyli na taśmie magnetofonowej, kasetce microdrive'u czy też dysku elastycznym. Informacje zapisane w pamięci zewnętrznej za pomocą programu edytora mogą być ponownie wprowadzone do pamięci edytora tekstu. Prócz tego teksty stworzone przez użytkowników da się połączyć ze sobą (w miarę wolnego miejsca w pamięci). Jedynym warunkiem łączenia pisanych za pomocą

edytora tekstów jest to, że ich sumaryczna długość nie może przekraczać 320 linii.

Jako, że praktycznie w naszym kraju nie istnieją ogólne sieci komputerowe, gdzie dostęp do poszczególnych jednostek sieci może odbywać się za pomocą linii telefonicznych, każdy tekst pisany jest w edytorze z myślą o jego przyszłym wydruku na drukarce. W związku z tym przed zapisaniem zawartości edytora w pamięci zewnętrznej trzeba pomyśleć o szacie graficznej zapisywanych danych. Przez odpowiednią szatę graficzną, prócz estetycznego wyglądu, rozumie się tu liczbę wierszy na stronie oraz wybór odpowiedniego kroju liter wydruku całości tekstu lub jego wybranych fragmentów. O tych wszystkich życzeniach program musi być poinformowany. Do tego celu służą znaki graficzne ZX Spectrum, po umieszczeniu ich w tekście są one interpretowane jako znaki sterujące drukarką (oprócz współpracy z ZX Printer) oraz są pomijane na wydruku. Znaczenie poszczególnych znaków może określić sam użytkownik.

Program *Tasword* jest przystosowany (firmowo) do współpracy z drukarką Epson FX 80 (lub Epson MX 80). Natomiast program *Procesor/Edytor tekstów* w swojej wersji podstawowej jest przystosowany do współpracy z drukarką D100 i interfejsem PB-3 o standardzie Centronics. Pracując z tą drukarką można korzystać między innymi z następujących możliwości:
— drukowania znaków dwukrotnie powiększonych,
— drukowania znaków o podwójnej wysokości,
— drukowania znaków o podwójnej szerokości,
— druku zagęszczonego,
— druku podwojonego z przesunięciem,
— ustalenia końca strony, itp.

Należy powiedzieć, że ułożenie liter oraz cyfr na klawiaturze mikrokomputera różni się od klawiatur polskich maszyn do pisania. Różnice są następujące:
— litery Z oraz X są zamienione miejscami,
— polskich liter można używać wciskając klawisz Q (pełniący rolę klawisza funkcyjnego) oraz a dla ą, e dla ę, itd.

Era mikrokomputerowych edytorów tekstów pozwala na skonstrouwanie układów klawiatur pozwalających na szybsze, niż dotychczas, pisanie. Obecnie próby takie prowadzone są w Wielkiej Brytanii i USA. Należy więc się zastanowić: czy wdrażając edytory tekstów nie skonstruować dla nich klawiatur z odpowiednim układem klawiszy i czy nie uczyć ich wykorzystania. Edytory tekstów nieuchronnie wypierają mechaniczne i elektryczne maszyny do pisania.

Na zakończenie kilka uwag praktycznych. Jeżeli program *Procesor/edytor* znajduje się w fazie obsługi magnetofonu lub drukarki, to można go przerwać za pomocą klawisza BREAK. Współpracę z edytorem wznawia się zachowując wprowadzony tekst za pomocą instrukcji GOTO 10. W takim przypadku może dojść jednak do skasowania tekstu, choć zdarza się to bardzo rzadko. Bardzo istotną wadą tego programu jest błędne interpretowanie polskich liter lub dyrektyw wprowadzanych przy użyciu klawisza Q wciśniętego z innym odpowiednim klawiszem. Najprawdopodobniej źle działają procedury obsługi klawiatury.

Szczególnie kłopotliwe jest zinterpretowanie znaku lub dyrektywy wprowadzanej przy użyciu Q jako dyrektywy kasowania tekstu (QV). Wprawdzie jej wykonanie wymaga potwierdzenia, lecz w czasie szybkiego pisania użytkownik może przypadkiem wcisnąć klawisz ,,t'' (tak) potwierdzający żądanie skasowania wszystkich danych. Opisane błędy zdarzają się rzadko, lecz ze względu na ich uciążliwość proponuje się częste zapamiętywanie kolejnych partii tekstu w pamięci zewnętrznej. Kolejną wadą jest zmiana sposobu wprowadzania polskich liter w fazie wyszukiwania wyrazów.

* Bazy danych 9.2

Komputery są niezwykle pomocne przy gromadzeniu, porządkowaniu i wyszukiwaniu informacji. Zostały do tego celu zaprzężone ,,od zarania'' i służą wiernie po dzień dzisiejszy. Trudno nawet wyobrazić sobie operowanie dużą ilością informacji bez pomocy maszyn cyfrowych. Komputerowe bazy danych znalazły zastosowanie w systemach rezerwacji biletów, komputerowych katalogach, np.: części w magazynie, asortymentu produktów wykonywanych przez fabrykę, danych personalnych itd.

Do realizacji baz danych potrzeba zwykle dużych systemów komputerowych wyposażonych w pojemną i szybką pamięć dyskową, pracujących w trybie wielodostępnym*) lub w sieci komputerowej.

ZX Spectrum nie nadaje się zbytnio do gromadzenia dużych zbiorów informacji ze względu na brak szybkiej pamięci zewnętrznej sterowanej z komputera, co nie znaczy, że jest do tego celu zupełnie nieprzydatne. Na przykład, kasetka microdrive'u ma zbyt małą pojemność (ok. 90 kbajtów), a samo urządzenie działa za wolno, lecz budowa bazy danych, w której informacje są gromadzone na kasetce, jest możliwa. Duże zbiory zajmować mogą wiele kasetek. Posiadacz ZX Spectrum wyposażonego w stacje dysków elastycznych mogą pokusić się o zrealizowanie lepszych (większych i szybszych) zbiorów danych. Można także budować bazę danych wykorzystując jedynie pamięć RAM komputera. Zbiór danych nie może być w takim przypadku zbyt duży. Jednak może on być z powodzeniem wykorzystany do tworzenia spisów adresów, katalogów bibliotecznych, danych inwentarzowych, spisów towaru w sklepie, ocen szkolnych itd.

Omówione zostaną najbardziej znane programy opracowane dla ZX Spectrum: *VU File* firmy Psion oraz *Masterfile* firmy Campbell Systems. Są one uniwersalnymi programami obsługi baz danych.

Zanim zostanie to dokładniej wyjaśnione, kilka ogólnych stwierdzeń o tych programach dla osób, które po raz pierwszy zetkną się z oprogramowaniem tego typu. Program obsługi bazy danych zajmuje się gromadzeniem danych w pamięci

*) Oznacza to, że z komputera może korzystać wielu użytkowników jednocześnie, jest on wyposażony w kilka terminali — dalekopisów lub monitorów z klawiaturą.

(w tym przypadku jedynie w pamięci RAM), przeszukiwaniem oraz drukowaniem informacji na ekranie lub drukarce w pewnym formacie. Program obsługi bazy danych współpracuje z pamięcia zewnętrzną, magnetofonem lub microdrive'm. Teoretycznie zestaw danych może mieć nieograniczoną długość. Może być porcjami ładowany z taśmy do pamięci i tak przetwarzany. W praktyce jest to dosyć niewygodne.

Bazę danych utworzoną przez program *VU File* lub *Masterfile* można wyobrazić sobie na podobieństwo zbioru zapisanych kartek (takie bazy danych nazywane są niekiedy systemami kartotek). „Kartki", z których składa się zbiór, nazywa się r e k o r d a m i. Zbiór rekordów nosi nazwę p l i k u. W czasie pracy programu cały plik dostępny dla obróbki musi się znajdować w pamięci RAM i dlatego jego wielkość jest ograniczona. W komputerach, które mają np. pamięć dyskową plik może być zapisany na dysku, dzięki czemu może zawierać znacznie więcej rekordów.

Programy *VU File* oraz *Masterfile* zostały napisane niemal w całości w języku asemblera po to, by zwiększyć szybkość pracy oraz pozostawić jak najwięcej pamięci na plik. *Masterfile* pozostawia danym 32 kbajty, *VU File* — 34 kbajty. Chociaż wydaje się, że to dużo, to jednak pamięć jest najbardziej istotnym ograniczeniem możliwości tych programów, gdyż plik może składać się co najwyżej z kilkudziesięciu rekordów (liczba rekordów zależy oczywiście od długości pojedynczego rekordu).

Rekordy nie mają na początku ustalonej struktury. Dlatego opisywane programy nazywano uniwersalnymi. Zależnie od potrzeb, użytkownik w czasie tworzenia bazy danych programuje jak ma być zbudowana „kartka" — czyli jakie pola powinny być umieszczone w rekordzie. Wszystkie rekordy pliku mają identyczną strukturę. W pola kolejnych rekordów wpisuje się dane.

Wyobraźmy sobie plik jako spis płyt w płytotece. Plik ten zawiera tyle rekordów, ile jest płyt w kolekcji. W każdym rekordzie należy zdefiniować pola, którym nadaje się nazwy np.:

- Autor
- Wykonawca
- Wytwórnia
- Symbol płyty
- Spis utworów

Każde takie pole ma swój identyfikator. Może on być pojedynczym znakiem — literą lub cyfrą, np. A: Autor, gdzie A jest identyfikatorem pola Autor.

Gdy struktura rekordu jest ustalona to rozpoczyna się tworzenie bazy danych. W pola rekordu wpisuje się dane — zazwyczaj tekst. Gotowy plik może być zapamiętany na kasecie magnetofonowej. Można też go przeszukiwać, dopisywać nowe informacje, usuwać niepotrzebne rekordy itd. Jeśli zbiór danych powinien zawierać więcej informacji, niż można zmieścić w pamięci, to należy zbudować kilka plików o identycznej strukturze.

Obróbka danych, np. wyszukiwanie, może przebiegać w ten sposób, że do pamięci ładuje się kolejne pliki z taśmy i na nich wykonuje potrzebne operacje (tylko na aktualnie załadowanym pliku).

Program *VU File* ustępuje znacznie swemu konkurentowi pod względem możliwości i komfortu pracy. Jest za to prostszy w użyciu. Plik *VU File* składa się z rekordów, które mają strukturę podobną do organizacji ekranu ZX Spectrum: 20 pól (wierszy) po 32 znaki każdy. Program *VU File* pozwala na określenie stałych elementów rekordu, np. tekstów, które będą pojawiały się na ekranie w czasie wyświetlania każdego rekordu, niezależnie od informacji, które zawarte są w rekordzie (np. nazwy pól).

Tworzenie zbioru danych odbywa się zwykle w ten sposób, że najpierw ustala się teksty stałe (tzw. raport), które określają postać w jakiej rekord pojawia się na ekranie. Ustalić można dodatkowy raport wydruku, czyli sposób w jaki dane drukowane są na drukarce (w jakim miejscu dane pole, jakim tekstem ma być poprzedzone, itd.). Następnie w tzw. „trybie zleceń" można wprowadzać rekordy. Komputer wyświetla teksty stałe, a użytkownik dopisuje w określone miejsca dane. Przypomina to wypełnianie formularza.

Tryb zleceń pozwala także na przetwarzanie pliku. Można przeglądać (na ekranie) kolejne rekordy, dodać lub usunąć rekord, wydrukować dowolny rekord na drukarce oraz posortować plik w porządku leksykograficznym danych zawartych w dowolnym polu rekordu. Można także wyszukiwać rekordy, które zawierają konkretne informacje, np. w pliku „Płytoteka" wyszukać można wszystkie płyty Bacha, albo płyty wydane przez wytwórnie CBS.

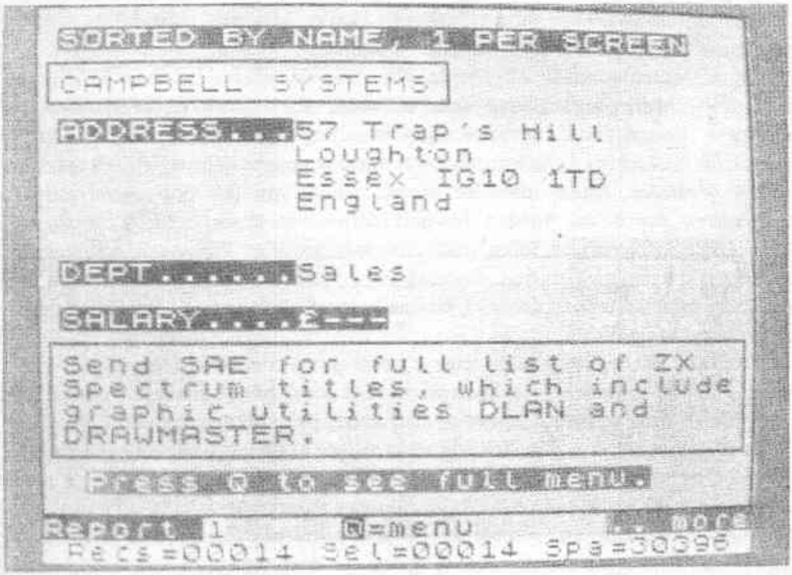
Wadą *VU File* jest to, że pole może zawierać co najwyżej 32 znaki, lecz mógłby on zostać uznany za całkiem niezły produkt, gdyby nie było programu *Masterfile*.

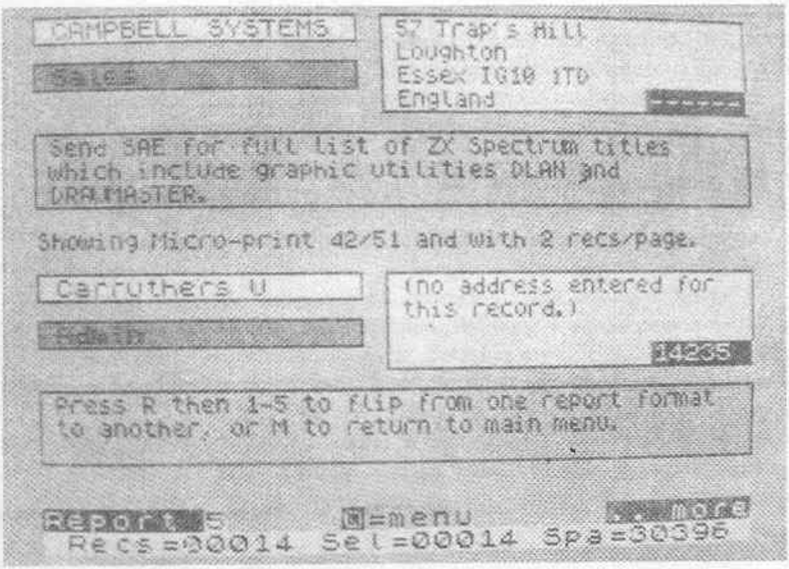
Masterfile jest nieco bardziej skomplikowany w obsłudze (takie wrażenie można odnieść na początku), lecz warto zadać sobie trochę trudu i opanować umiejętność posługiwania się nim. Gdyby autorzy, wzorem zachodnich czasopism komputerowych, oceniali programy daliby mu piątkę.

Program ten współpracuje z użytkownikiem w taki sposób, by w jak największym stopniu ułatwić mu korzystanie z systemu. *Masterfile* na początku drukuje na ekranie tzw. menu główne (ang. *Main Menu — MM*), czyli spis kilkunastu opcji — możliwości do wyboru. Wciśnięcie odpowiedniego klawisza powoduje wybranie opcji, np. funkcji dodawania nowego rekordu do pliku. Często użytkownik ma wtedy dalsze opcje do wyboru, gdyż na ekranie drukowane są kolejne menu. Niekiedy komputer podpowiada, co należy uczynić dalej.

Podczas wyświetlania głównego menu (MM) oraz w kilku innych przypadkach na ekranie wydrukowane są informacje o liczbie rekordów w pliku, wielkości wolnej pamięci oraz liczbie rekordów wybranych z pliku do dalszego przetwarzania. Plik utworzony przez program *Masterfile* może zostać zapamiętany na kasecie magnetofonowej lub kasecie microdrive'u, a następnie wczytany w celu przejrzenia, aktualizacji, itd. Struktura rekordów jest programowana. Rekord składa się z pól (maks. 26*)) o długości do 128 znaków każde, mogą one mieć nazwy. Wielkość pamięci zajmowanej przez rekord zależy od długości wpisanych w nim danych. Pola mogą mieć nazwy, określają one strukturę rekordu.

*) Drobna poprawka programu pozwala zmienić maksymalną liczbę pól.


SORTED BY NAME, 1 PER SCREEN
CAMPBELL SYSTEMS
ADDRESS....57 Trap's Hill
Loughton
Essex IG10 1TD
England
DEPT......Sales
SALARY....£---
Send SAE for full list of ZX Spectrum titles, which include graphic utilities DLAN and DRAWMASTER.
Press Q to see full menu.
Report 1
M=menu
... more
Recs=00014 Sel=00014 Spa=30396


CAMPBELL SYSTEMS
57 Trap's Hill
Loughton
Essex IG10 1TD
England
Send SAE for full list of ZX Spectrum titles which include graphic utilities DLAN and DRAWMASTER.
Showing Micro-print 42/51 and with 2 recs/page.
Carruthers U
(no address entered for this record.)
14235
Press R then 1-5 to flip from one report format to another, or M to return to main menu.
Report 5
M=menu
... more
Recs=00014 Sel=00014 Spa=30396

Można definiować także raporty rekordu — sposób w jaki rekord jest obrazowany na ekranie telewizyjnym lub drukarce.

W odróżnieniu od *VU File*, *Masterfile* pozwala zdefiniować do 36 różnych formatów. Można przeglądać plik używając dowolnego ze zdefiniowanych raportów. Forma graficzna raportu jest bogata. Programuje się miejsca ekranu, w których drukowane będą informacje zawarte w poszczególnych polach rekordu, kolory wydruku. Dane oddzielać można pionowymi lub poziomymi liniami, zamykać w ramki itd. Można również definiować raporty, które na ekranie wyświetlają informacje z kilku rekordów jednocześnie. Procedura definiowania raportu jest żmudna i trudna. Chociaż jest to praca wykonywana zwykle tylko raz (przy tworzeniu bazy danych), można było zorganizować to lepiej, niż zrobili to autorzy *Masterfile'a*.

Program *Masterfile* współpracuje z programem o nazwie *Micro-Print 42/51*, który pozwala drukować w jednym wierszu 42 znaki lub 51 znaków (zamiast 32).

Masterfile ma wygodny i sprawny mechanizm przeszukiwania danych. Program może szybko wybrać z pliku rekordy zawierające aktualnie potrzebne informacje. W zbiorze wybranych rekordów można prowadzić dalszą selekcję, np. z pliku „Płytoteka" można wybrać rekordy zawierające dane o płytach Chopina (autor — Chopin). Z płyt Chopina można wybrać nagrane przez CBS, z tych płyty Horowitz'a itd. *Masterfile* dzieli rekordy pliku na dwie kategorie: wybrane i niewybrane. Rekordy wybrane są dostępne do dalszego przetwarzania. Wciśnięcie pojedynczego klawisza pozwala wszystkie rekordy uznać za wybrane. Komputer przeszukuje plik i rekordy spełniające określony warunek, np. zawierające określone informacje, zaznaczane są jako wybrane. Można prowadzić poszukiwania wśród wszystkich rekordów, rekordów aktualnie wybranych lub niewybranych. *Masterfile* oblicza sumę oraz wartość średnią określonych pól wybranych rekordów, jeśli zawierają one dane numeryczne. Możliwości przeszukiwania danych zależą od sposobu zdefiniowania rekordu.

Program ten stwarza także możliwość obróbki danych za pomocą programu użytkownika napisanego w języku Basic. Zdefiniować można cztery podprogramy, które będą wywoływane:

— na początku przetwarzania pliku,
— na początku przetwarzania każdego wybranego rekordu,
— na początku każdego z pól każdego rekordu,
— na końcu rekordu.

Wstawki w języku Basic dołączone do programu *Masterfile* pozwalają np. stworzyć nową pozycję w rekordzie, obliczać różne wartości na podstawie danych zawartych w pliku, itd. Na przykład, gdy plik zawiera spis towarów w sklepie, a każdy rekord zawiera: nazwę towaru, cenę i ilość, to opisany mechanizm pozwala obliczyć np. wartość towarów w sklepie.

Program *Masterfile* współpracuje z microdrive'm, jednak nie zapewnia to zwiększenia pamięci przeznaczonej na plik, pozwala jedynie na skrócenie czasu ładowania pliku do pamięci lub zapamiętywania go na kasetce.

Na rynku znajduje się kilka wersji programu. Wersje o numerach od 8 w górę zawierają program *Micro-Print 42/51*. *Masterfile* przypomina profesjonalne

programy obsługi baz danych. Może stanowić nieocenioną pomoc przy gromadzeniu i przetwarzaniu zbiorów danych. Jego zastosowanie ogranicza jedynie wielkość pliku. Jednak mimo to jest on godny polecenia.

Polską wersją *Masterfile ver. 09* jest program *Baza danych* (np. wersja 03). Zajmuje on 1000 bajtów więcej niż pierwowzór. Niefortunnie zmieniono sposób wybierania rekordów. Program ten pozwala na użycie polskich liter, lecz mechanizm ten działa niepoprawnie. Z tych względów należy polecić oryginał, a nie jego polską wersję mimo, że jest ona łatwiejsza w użyciu, ponieważ nazwy opcji i komunikaty drukowane są po polsku.

* Programy kalkukacyjne 9.3

Na całym świecie żadne szanujące się biuro nie może obyć się bez sporządzania bilansów, zestawień, planów, list płac, itd. Niejednokrotnie prawidłowe sporządzenie takich zestawień jest podstawą oceny działania całego przedsiębiorstwa. Czy tak prosty komputer jak ZX Spectrum jest w stanie ulżyć w pracy działów finansowych, księgowości? O ile zagadnienia nie są zbyt duże, okazuje się, że tak.

Są firmy sprzedające oprogramowanie, które wręcz specjalizują się w tworzeniu programów pomocnych w „sprawozdawczości" i to począwszy od wielkich komputerów, do tych całkiem, jak Spectrum, małych. Program sprzedany na świecie (1984 r.) w największej liczbie kopii to właśnie program kalkulacyjny — *VISI-CALC* firmy Visicorp dla Apple II. Firma zarobiła wiele, ale ci co go kupili, też na tym nie stracili, bowiem kalkulacje kosztów produkcji potrafili przeprowadzić szybciej i dokładniej niż konkurencja. Najnowsze programy tego typu potrafią nie tylko tworzyć tabelaryczne zestawienia, ale też przedstawiać wyniki w postaci wykresów kołowych, histogramów, stawiać prognozy ekonomiczne opierając się na modelach rynku.

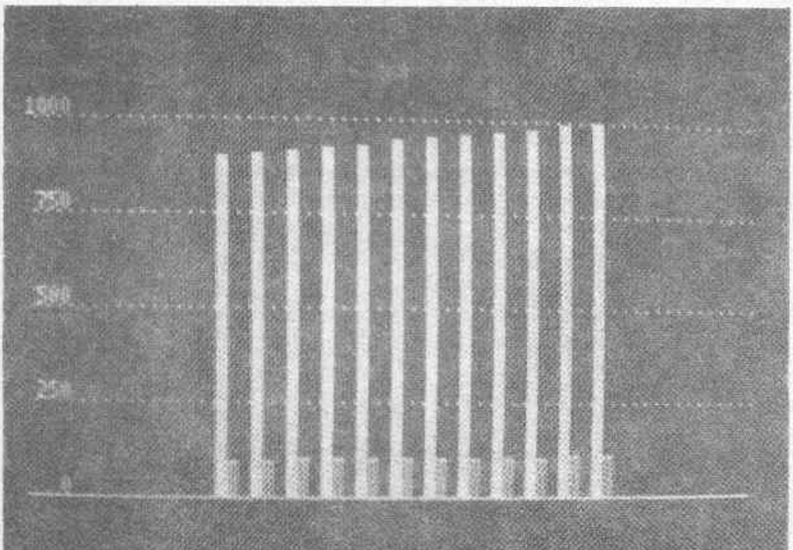
Edytor tekstów, baza danych, program kalkulacyjny i komputer to obecnie podstawowe narzędzia pracy dla personelu biura — jeszcze nie wszędzie, ale to tylko kwestia czasu.

Dlaczego programy kalkulacyjne*) są tak przydatne? Jak się ich używa? Zwykle gdy trzeba przeprowadzić planowanie, np.: listy płac, siada się z ołówkiem w ręku nad kartką papieru. Dzieli się ją na wiele wierszy i kolumn tworząc tabelę — lista pracowników na poszczególne miesiące. Gdzieś na marginesie pisze się dane wejściowe: ile procent dochodu można przeznaczyć na płace, ile z tego zabiorą podatki, o ile więcej trzeba zapłacić kierownikowi, kto pracuje w warunkach szkodliwych, itd. Mozolnie wpisując w kolejne kratki liczby, procenty i ułamki próbuje się bilansować wydatki. Gdy pierwsza koncepcja okaże się do niczego, idzie w ruch gumka i znów planuje się, wylicza i tak jeszcze długo, aż osiągnie się końcowy, zadowalający pracodawcę i pracownika efekt.

*) Ang. *spreadsheet*.

Metoda działania programu kalkulacyjnego jest taka sama. I tu mamy do czynienia z kartką papieru (w pamięci komputera) podzieloną na poszczególne pola o określonych współrzędnych. Zwykle dostępna tabela jest tak duża, że nie mieści się na raz na ekranie — widać tylko pewien jej fragment, który można przesuwać po całej, dużej „kartce". W nagłówki wierszy i kolumn można wpisywać z klawiatury ich oznaczenia, a więc jak w przykładzie: nazwiska i nazwy miesięcy. Ale obliczeń nie trzeba wykonywać samemu — wystarczy podać wzór, np. jak obliczać pensję Kowalskiego: 1/20 funduszu A+1500 dodatku + każdego miesiąca o 100 więcej — stąd wiadomo, ile trzeba mu będzie zapłacić od stycznia do grudnia, ile w sumie wyniesie podatek itd. Zmiana jednego ze składników nie będzie tu oznaczać konieczności wycierania i poprawiania wyników — przeliczenia dokonuje się pojedynczą komendą! Oczywiście uzyskane wyniki można przenieść na papier. Na dołączonej drukarce wydrukować tabelę, wykres czy też przesłać je (np. przez telefon) do centrali firmy. Taki program przydaje się nie tylko ekonomistom, ale też i inżynierom — wzory obliczeń mogą być naprawdę bardzo skomplikowane.

Te wszystkie opisane pokrótce możliwości realizują programy kalkulacyjne dostępne na ZX Spectrum. Jest ich kilka. Autorzy mieli możliwość używania programu *VU-CALC* firmy Psion i *Omnicalc-2* firmy Microsphere. Możliwości tych programów są bardzo duże, lecz niewątpliwie nowy *Omnicalc* ma przewagę nad swoim konkurentem. W obu z nich użytkownik posługuje się tabelą, której fragment (4 kolumny × 18 wierszy *VU-CALC*, 3 × 16 — *OM2*) jest stale widoczny na ekranie. Jej maksymalny rozmiar wynosi dla *Omnicalc*: 250 wierszy × 99 kolumn, a dla *VU-CALC*: 60 × 60 (w *Omnicalc* istnieje też wymiar minimalny 15 × 3).



W pojedynczym polu mieści się maksimum 7 znaków (*Omnicalc* przyjmuje tylko duże litery). Oczywiście liczby, czy teksty mogą być dłuższe — zajmują wtedy więcej niż jedno pole. Każde z pól może zawierać liczbę, tekst, bądź wartość wyrażenia, zależnego od wartości wpisanych w pola określone innymi współrzędnymi. Może też być puste. Wyrażenia mogą zawierać nie tylko operatory arytmetyczno-logiczne, lecz także funkcje (*Omnicalc* wszystkie dostępne z Basica). Wzory mogą więc być bardzo skomplikowane, służyć nie tylko do obliczania list płac, ale także do obliczeń inżynierskich, do modelowania pewnych reguł rynkowych, m.in. przez możliwość użycia funkcji RND, a więc elementu losowości.

Ważną cechą wprowadzanych wyrażeń jest ich względność, tzn. jeśli potrzeba, aby polu o współrzędnych c1 (c — trzeci wiersz, 1 — pierwsza kolumna) została nadana wartość sumy pól a1 i b1, wprowadza się wzór a1 + b1. Jeżeli teraz chcielibyśmy nadać polu c2 wartość a2 + b2 wystarczy skorzystać z funkcji Repeat, która powieli wyrażenie na żądany wymiar tablicy. Oczywiście wybrane pola można traktować w obliczeniach jako stałe (oznaczane odpowiednio: ($,k)).

Liczby można ze sobą łączyć, tzn. traktować w obliczeniach jedno z pól jako część bardziej znaczącą liczby, a następne jako mniej znaczące itd. Jest to ważne przy obliczeniach dużych kwot z dokładnością np. do 1 gr.

Wydruk w pojedynczym polu może być formatowany, tzn. przedstawiany w postaci całkowitej, bądź z dokładnością do drugiego miejsca po przecinku, a w przypadku *VU-CALC* również tabulowany lewo- lub prawostronnie. Wyrażenia według których oblicza się wartość pól są zapamiętywane. Umożliwia to szybką ich modyfikację — przeliczenie całej tabeli dla zmienionych danych wejściowych odbywa się jednym rozkazem (C—*calculate*). Kalkulacje można prowadzić raz, bądź dopóty, dopóki w któreś z pól nie wpisze się zera (*Omnicalc*). Obliczenia są szybkie, o wiele szybsze niż wykonywane ręcznie.

Jak już wspomniano, na ekranie widać w danej chwili tylko wycinek całego zestawienia. Przesuwanie się po nim odbywa się tak jak w Basicu za pomocą kursora, bądź przez zażądanie ustawienia się w miejscu o zadanych współrzędnych. Aby ułatwić pracę użytkownikom postarano się też o instrukcję wymiany całych wierszy i kolumn (T w *VU-CALC*, poprzez tzw. Workarea w *Omnicalc*).

Gdy trzeba przerwać obliczenia, to ich wyniki można zapamiętać na taśmie magnetofonowej (SAVE nazwa; w *Omnicalc* istnieje też VERIFY). Następnego dnia pracę wystarczy rozpocząć od LOAD, aby wczorajszy efekt powrócił na ekran.

Gdy wreszcie zbudowana zostanie tabela zaspokajająca potrzeby użytkownika, można ją wydrukować na dołączonej drukarce. *VU-CALC* pozwala przesłać wyniki pracy tylko na drukarkę ZX Printer. *OM2* uwzględnia współpracę z innymi drukarkami, obsługiwanymi przez Interface 1 lub przez własny program użytkownika. Dzięki współpracy z Interface 1 *Omnicalc 2* pozwala przesłać wyliczone wartości na microdrive lub przez łącze RS 232 np.: do modemu, a także w sieć lokalną.

Omnicalc 2 ma jeszcze jedną niezwykle przydatną funkcję — tworzenie wykresów (histogramów) wartości zawartych w poszczególnych polach macierzy.

Naraz można przedstawić graficznie do 3 grup maksymalnie po 60 wartości. Na przykład pensje trzech pracowników w ciągu 12 miesięcy zostaną przedstawione w postaci 3 różno-zaczernionych prążków dla każdego miesiąca. Słupki te mogą też stać jeden na drugim. Histogram wyliczany jest dla wartości z zakresu ±3 mld, a więc nawet duże przedsiębiorstwa mogą zaprezentować wyniki swojej pracy w estetycznej formie. Zaprezentować, bowiem wykresy można opisać i wydrukować na dołączonej drukarce (oczywiście o możliwościach graficznych).

Omnicalc ma też możliwość wprowadzania nazw miesięcy jednym naciśnięciem klawisza, a także sumowania wartości w kolejnych wierszach i kolumnach. Sygnalizuje też specjalnym dźwiękiem i napisem powstanie błędu. Istnieje też wersja *OM 64000*, która pozostawia ostatnie 1500 bajtów pamięci wolne, przeznaczone na asemblerowy program użytkownika, np. obsługi drukarki czy plotera.

Autorzy uważają, że programy kalkulacyjne na ZX Spectrum można z całą odpowiedzialnością polecić małym i średnim firmom. Programy takie jak *Omnicalc 2* są naprawdę bardzo proste w obsłudze — nawet kilkugodzinna nauka wystarcza osobom mającym mało styczności ze sprzętem komputerowym. Tworzenie kosztorysów, kalkulacji jest często czynnością mechaniczną, polegającą na zmianie parametrów raz stworzonego modelu. Przy użyciu komputera czas wykonania takiej pracy liczy się w godzinach, a nie dniach. Gdy program kalkulacyjny sprzęgnięty jest z graficznym, wyniki jego działania są czytelne nawet dla laika, a i dla specjalisty są łatwiejsze w interpretacji.

* Programy graficzne 9.4

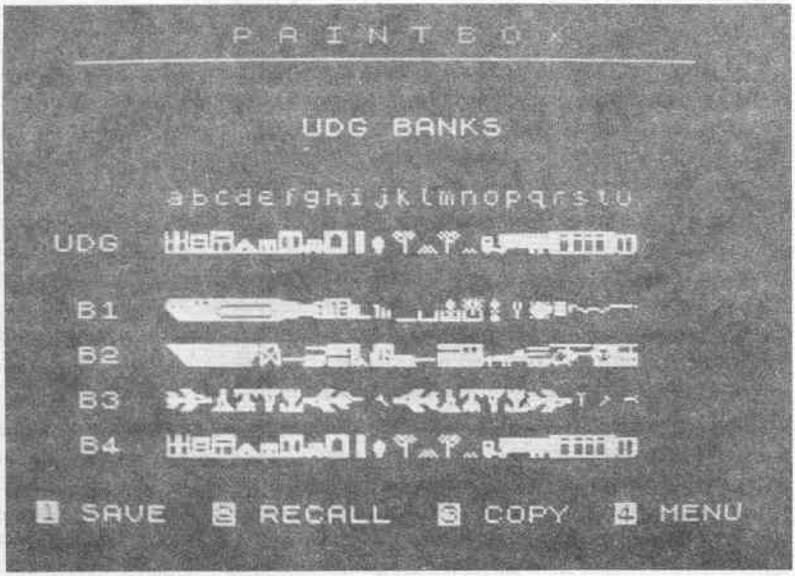
Grafika jest nieodzownym elementem zdecydowanej większości programów. Ma ona szczególne znaczenie, gdy obraz na ekranie telewizora ma oddziaływać na wyobraźnię użytkownika mikrokomputera. Zaprojektowanie interesującej grafiki nie jest wcale proste. Jest za to bardzo pracochłonne. Przy rozwiązywaniu problemów związanych z projektowaniem obrazów pomocne są programy graficzne, czyli programy rozszerzające możliwości graficzne ZX Spectrum, lub ułatwiające tworzenie grafiki. Narzędzi programowych tego typu jest bardzo dużo, wśród nich znajdują się między innymi takie, jak: *Artist*, *VU-3D*, *Melbourne Draw*, *Paintbox*, *Screen Machine*, *Spectrum Sprites*, *Pencil Designers*, *Leonardo* czy *Dlan*. Część z tych programów ma swoje polskie odpowiedniki. Jednakże, nie warto o nich mówić. Ich „polskość" często sprowadza się do zmiany nazwy przetłumaczenia na język polski dostępnych opcji (o ile są napisane w Basicowej części programu) oraz dołączenia „wizytówki" firmy, która go sprzedaje. Aby nie być gołosłownym, np.: program *VU-3D* jest sprzedawany przez jedną z firm pod nazwą „*Cube I*".

Programy graficzne rozszerzają możliwości ZX Spectrum przez:

— dołączanie programu do projektowania grafik definiowanych przez użytkownika (UDG) oraz dowolnych elementów generatora znaków np.: *Paintbox*, *Melbourne Draw*, i in.,

— przypisanie instrukcji graficznych do poszczególnych klawiszy np.: *VU-3D*, *Melbourne Draw*,
— wprowadzanie nowych poleceń np.: *Pencil Designers*, czy też wcześniej omówiony MegaBasic.

Do czego wykorzystywać graficzne narzędzia programowe? Najbardziej klasycznym przykładem jest projektowanie własnych grafik definiowanych lub też własnego generatora znaków. Wykonanie takiej pracy na papierze jest żmudne i pracochłonne. Natomiast programy graficzne pracę tę znacznie ułatwiają. I tak, program *Paintbox*, między innymi, daje możliwość zdefiniowania 84 własnych znaków graficznych — UDG. Biorąc pod uwagę opcję INVERSE (patrz rozdz. 3) dostępną z poziomu programu, w praktyce możliwe jest zdefiniowanie 168 symboli UDG.



Konstruowanie każdego z symboli odbywa się na odpowiednio powiększonej przygotowanej do tego celu matrycy 8 × 8 punktów. Proces ten jest bardzo prosty. Ponieważ obszar UDG pozwala na jednorazowe określenie 21 symboli, to wszystkie 84 symbole przechowywane są w tzw. czterech bankach. Dostęp do wybranego z banków odbywa się przez zmianę adresu UDG, co w praktyce odbywa się przez naciśnięcie odpowiedniego klawisza. Poza tym możliwe jest natychmiastowe wykonanie lustrzanego odbicia danego znaku z lewa na prawo, a także wykonalne są obroty o kąt 90° każdego ze zdefiniowanych znaków.

Nowo zaprojektowane obszary grafik definiowanych można zapisać na taśmę i wykorzystywać je do tworzenia skomplikowanych obrazów. Nie są to jedyne rozszerzenia, które wprowadza *Paintbox*. Z poziomu programu można kreślić

także dowolne łuki, a każdy zamknięty krzywymi obszar może zostać zamalowany (wypełniony kolorem) itd.

Kolejnym ciekawym programem graficznym jest *Melbourne Draw*. Ma on kilka trybów pracy takich jak: projektowanie UDG, tryb przesuwania „znacznika" bez zostawiania śladu, rysowania znacznikiem poruszanym wybranymi klawiszami, kasowanie narysowanych linii, tryb tekstowy, zmiany atrybutów, tryb „skala" itp. Dużą zaletą tego programu jest możliwość łączenia grafiki z tekstem. W trybie tekstowym możliwe jest umieszczanie napisów na ekranie telewizora w jednym z czterech kierunków, a więc od lewej do prawej, z góry na dół, z prawej do lewej (litery odwrócone) oraz z dołu w górę. Kierunek pisania wybiera się klawiszami sterowania kursorem zgodnie z ich oznaczeniem. Litery pisane na uprzednio narysowanych liniach niszczą je w obrębie pola 8 × 8 punktów. Aby tego uniknąć należy najpierw wprowadzić na ekran tekst, a później nanieść na nim odpowiedni rysunek.

W programie *Melbourne Draw* niebezpieczne jest korzystanie z trybu skala. W trybie tym możliwe jest zmniejszenie lub zwiększanie wymiarów rysunku o 1/7. Program przy zmianie skali traci część pierwotnych informacji. Powrót do początkowych wymiarów jest bardzo trudny, a często niemożliwy. Dlatego przed użyciem tego trybu zaleca się zapisanie oryginalnego rysunku na taśmę.

Program *Melbourne Draw* ma opcję „krata". Po naciśnięciu odpowiedniego klawisza na ekranie pojawia się szachownica składająca się z białych pól zdefiniowanych na przemian z różnym poziomem jasności (jedno pole o jasności normalnej, a obok jasności podwyższonej). Krata niczego nie zmienia. Natomiast znacznie ułatwia projektowanie rysunków oraz rozlokowanie napisów. Po wprowadzeniu kraty z ekranu znikają uprzednio nałożone atrybuty. Widoczny jest tylko kształt rysunku. Atrybuty określa się dopiero po usunięciu kraty z ekranu.

Do elementów grafiki należą także kroje liter i cyfr. Interesujące rozwiązanie tego problemu przedstawione jest w programie *Dlan*. Program ten pozwala na definiowanie dowolnych okien w obrębie całego ekranu (24 wiersze po 32 znaki w każdym). Wewnątrz wcześniej zdefiniowanego okna można wyświetlac dowolne napisy o dowolnych atrybutach (w miarę możliwości ZX Spectrum). Do wyświetlania napisów można użyć jedenastu krojów liter zdefiniowanych na stałe w programie. Poza tym napisy mogą być przesuwane w obrębie każdego określonego okna w jednym z dowolnych kierunków podstawowych (tzn. góra, dół, lewo, prawo). Jeżeli użytkownik ZX Spectrum chciałby w swoich programach wykorzystywać tylko litery o różnych wymiarach, to gotowe rozwiązanie znajduje się na kasecie demonstracyjnej. Procedura powiększania liter znajduje się prawie w każdym z jej programów. Zwykle jej początek znajduje się w linii o numerze 9300 i kończy się przez RETURN. Niezbędny do jej działania jest program w kodzie maszynowym umieszczany najczęściej za podstawowym blokiem w Basicu.

Pisząc o programach graficznych nie sposób nie wspomnieć o najstarszym z nich — o *VU-3D*. Jest to jeden z niewielu, napewno najlepszy wśród pozostałych, programów pozwalających na konstruowanie grafik przestrzennych. Projekto-



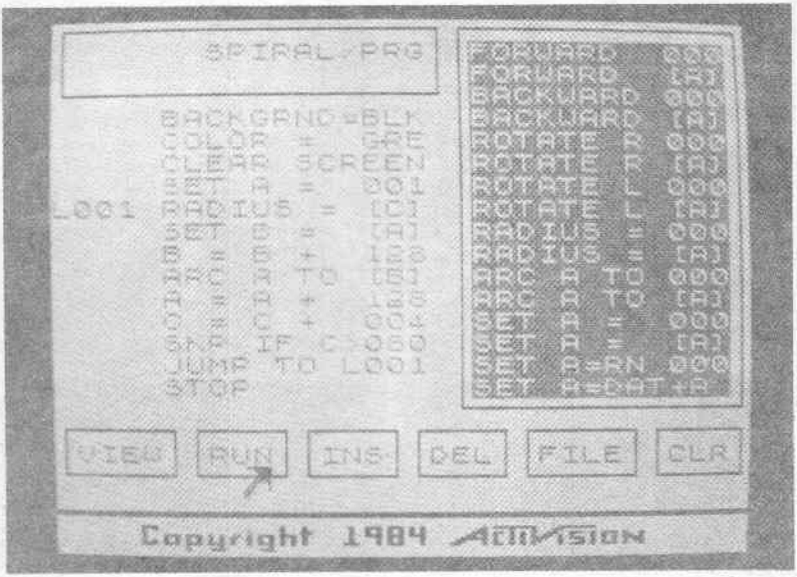
wany obiekt może być definiowany w postaci jego kolejnych przekrojów. Po zakończeniu fazy projektowania na narysowanej na ekranie bryle można wykonywać następujące czynności:

— obroty w przestrzeni,
— powiększanie lub zmniejszanie,
— zbliżanie i oddalanie z zachowaniem perspektywy,
— kasowanie niewidocznych krawędzi,
— kolorowanie,
— cieniowanie przez oświetlenie jej z odpowiednio ustawiqnych względem obiektu źródeł światła.

Jeżeli w trakcie oględzin projektu bryły użytkownik dostrzeże jakieś niedociągnięcia lub uchybienia, to istnieje szansa ich usunięcia po przejściu do trybu modyfikacji. Projekt bryły zapisany w pamięci komputera może być zapisany na taśmie magnetofonowej (lub na taśmie microdrive — po drobnej modyfikacji programu).

Rysunki otrzymane za pomocą *VU-3D* (i nie tylko) mają pewne niedociągnięcia związane z małą rozdzielczością grafiki ZX Spectrum — tylko 256 × 192 punkty. Przez to niektóre linie proste (idące pod kątem) mają kształt łamanych. Mimo tej wady program ma spore walory edukacyjne i z powodzeniem może być wykorzystywany na lekcjach geometrii, po uprzednim przygotowaniu projektów brył przez nauczyciela.

Inny od dotychczas omawianych programów jest *Pencil Designers* (PD). Projekty graficzne powstają w wyniku realizowanych programów zapisanych w języku, który wprowadza PD. Instrukcje PD naśladują trochę instrukcje graficzne Logo. Interesująca jest instrukcja pozwalająca na śledzenie zmian wartości



wybranej zmiennej podczas wykonywania się programu. Dzięki niej łatwiej poprawia się błędy, o których program informuje użytkownika własnymi komunikatami.

Pencil Designers ma narzędzia do definiowania procedur rekurencyjnych. Jednym z parametrów takiej procedury musi być z góry określony poziom rekursji, co zdaniem autorów jest wadą.

Do omawianego programu dołączonych jest 12 krótkich programów, które demonstrują możliwości programu i pozwalają na szybsze opanowanie techniki pisania programów w języku *Pencil Designers*. Wydruk jednego z nich wraz z całym ekranem przedstawiono poniżej. Jak widać, ekran podzielony jest na cztery części. W pierwszej, na górze ekranu, wyświetlana jest nazwa projektu i komunikaty o błędach. W oknie z prawej strony wyświetlone są nowe instrukcje wprowadzone przez PD. Dół ekranu przeznaczony jest na tzw. komendy podstawowe, jak: RUN, DELETE itd... Resztę miejsca zajmuje program użytkownika — tu demonstracyjny. Okna z programem i instrukcjami mogą być przewijane (ang. *scroll*). Każdy zaprojektowany za pomocą PD obraz można zapisać w pamięci zewnętrznej.

Do osobnej grupy programów graficznych należą programy umożliwiające projektowanie grafiki sprite'ów (patrz rozdz. 3 oraz rozdz. 7 MegaBasic). Jednym z nich jest *Sprite Project*. Umożliwia on definiowanie 16 sprite'ów o wymiarach 16×16 punktów. Określone sprite'y mogą poruszać się w dowolnym kierunku. Program ten rozszerza możliwości graficzne interpretera Basica. Ma on znacznie mniej możliwości od MegaBasicu opisanego wcześniej.



Ostatnio ukazały się na rynku programy *Leonardo* i *Artist*. Leonardo zyskał bardzo wysokie oceny w czasopiśmie *Sinclair User*. Należy zaufać ekspertom miesięcznika, autorzy bowiem nie mieli możliwości sprawdzenia go.

Artist przewyższa inne programy graficzne, Umożliwia tworzenie animowanych obrazów, zmianę zawartości generatora znaków, uży kowanie znaków różnej wielkości (do 64 znaków w wierszu), rysowanie "pędzlem" linii o zmiennej szerokości i różnej fakturze, wypełnianie obszarów ograniczonych krzywymi wybranym wzorem w jednym z używanych w Spectrum kolorze itd. Ponadto fragment tworzonego rysunku może być wyjęty z całości, powiększony, zmodyfikowany i wstawiony z powrotem. Pełniejszy opis tego programu można znaleźć w w nr 3 miesięcznika Komputer.

* Programy edukacyjne 9.5

Ostatnimi czasy szeroko dyskutowanym tematem stało się zastosowanie komputerów do wspomagania nauczania. Pomysł nie jest nowy, lecz dopiero wprowadzenie na rynek setek tysięcy bardzo tanich (jak na komputery i na warunki zachodnie) maszyn stworzyło realne możliwości jego realizacji na szeroką skalę. Przeróżne firmy produkujące oprogramowanie opracowały dziesiątki programów edukacyjnych dla ZX Spectrum. Ich liczba jest porównywalna nawet z liczbą gier.

Komputery wykorzystuje się do bardziej intensywnego nauczania. Uzyskuje się to przez:

— indywidualizację nauczania,
— aktywne angażowanie ucznia w ten proces.

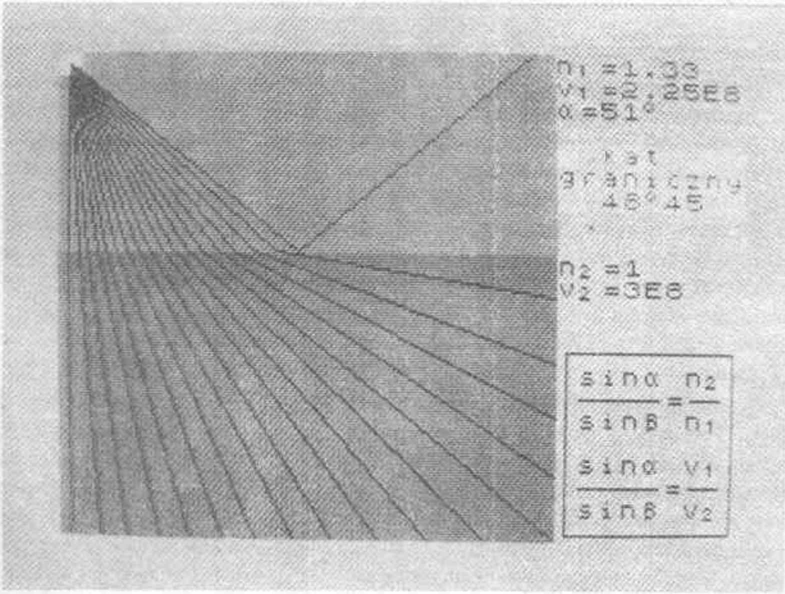
Komputer znakomicie nadaje się do nauczania, ponieważ kolorowe ruchome obrazy, które wytwarza na ekranie telewizyjnym, oddziałują na wyobraźnię i pamięć wzrokową ucznia. Wykład, który zręcznie wykorzystuje obrazy do ilustrowania pojęć abstrakcyjnych, toku rozumowania lub przebiegu procesu jest bardzo skuteczny, wiedza jest łatwiej przyswajana. Ważną cechą komputerowych programów edukacyjnych jest zdolność prowadzenia dialogu z uczniem. Komputer nie pozwala na bierność. Zadaje pytania. Reaguje na odpowiedzi ucznia. Musi on podejmować decyzje o dalszym przebiegu programu. Komputer ocenia stopień opanowania wiedzy, niekiedy nagradza ucznia. Czasem nagrodą jest krótka zabawa z komputerem, innym razem pochwała wydrukowana na ekranie. Nie ulega wątpliwości, że nauka aktywna jest bardziej efektywna od biernej.

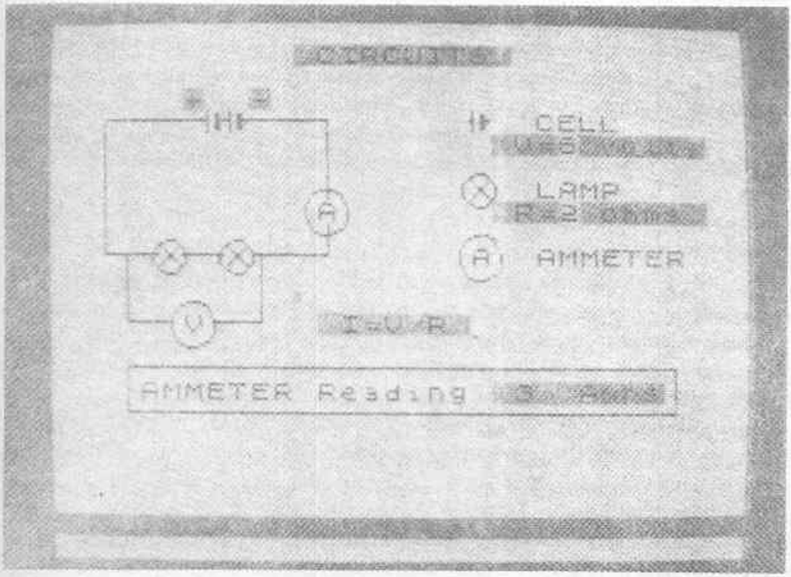
Programy edukacyjne to np. lekcje komputerowe, w których komputer uczy pewnego materiału: na ekranie drukuje treść wykładu, rysunki, tłumaczenia itd., i zwykle na zakończenie (albo też na bieżąco) sprawdza stopień opanowania wiadomości, powtarza pewne partie materiału, podsumowuje. Często spotyka się także komputerowe ćwiczenia (ang. *drill and practice*), testy i sprawdziany, gry o charakterze edukacyjnym — przeznaczone dla przedszkolaków oraz dzieci z młodszych klas szkoły podstawowej. Do nauczania stosuje się także symulatory, tzn. programy, które symulują przebieg różnych zjawisk fizycznych, czy procesów ekonomicznych lub społecznych, wykorzystując do tego celu ich matematyczne modele.

W polskich szkołach znajduje się kilka tysięcy komputerów. Z doświadczeń autorów wynika, że wielu prywatnych właścicieli tego komputera chciałoby uczyć za jego pomocą własne dzieci. Często sami piszą programy do nauki. Komputer może służyć dzieciom opóźnionym w nauce lub mniej zdolnym. Pomoże w rozwijaniu zainteresowań i zdobywaniu wiedzy.

Wybór programów edukacyjnych dla ZX Spectrum nie jest zbyt różnorodny. Zostały one w większości opracowane przez komercyjne firmy produkujące oprogramowanie. Niżej przedstawiono przykłady takich programów, które powinny zapoznać Czytelników z tego typu oprogramowaniem i zachęcić do jego stosowania.

Programy edukacyjne dla ZX Spectrum przeznaczone są dla szerokiego kręgu odbiorców: od przedszkolaków (np. *Counting* — zestaw trzech programów dla dzieci, uczący liczenia do dwudziestu) do ludzi dorosłych (np. komputerowy kurs asemblera Z80: *The Complete Machine Code Tutor*). Firma *Homestudy* opracowała komputerowe kursy matematyki i fizyki z zakresu tzw. Secondary school — co odpowiada klasom od szóstej do dziesiątej w polskich szkołach. Skomputeryzowano dwa kursy matematyki wg programu londyńskiego oraz Cambridge oraz program SCE — nauczania fizyki. Na pojedynczej kasecie znajduje się kilkanaście programów edukacyjnych, np. kurs matematyki wg programu Cambridge (*Cam-*


n1 =1.33
v1 =2.25E8
α =51°
kat
graniczny
48°45
n2 =1
v2 =3E8
sinα/sinβ = n2/n1
sinα/sinβ = v1/v2


CELL
LAMP
AMMETER
AMMETER Reading

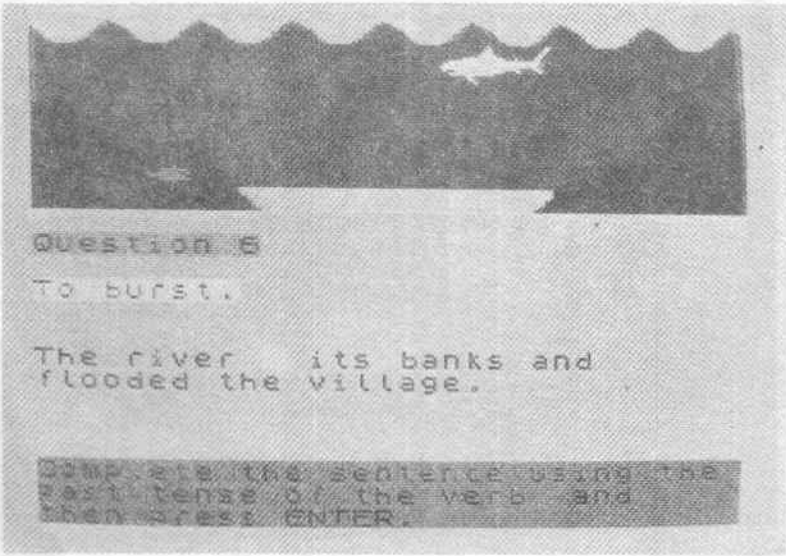
bridge Syllabus) składa się z 12 kaset, obejmuje m.in. ułamki, funkcje, układy równań, zbiory, macierze, prawdopodobieństwo, itd. Jedna z kaset zawiera pytania egzaminacyjne.

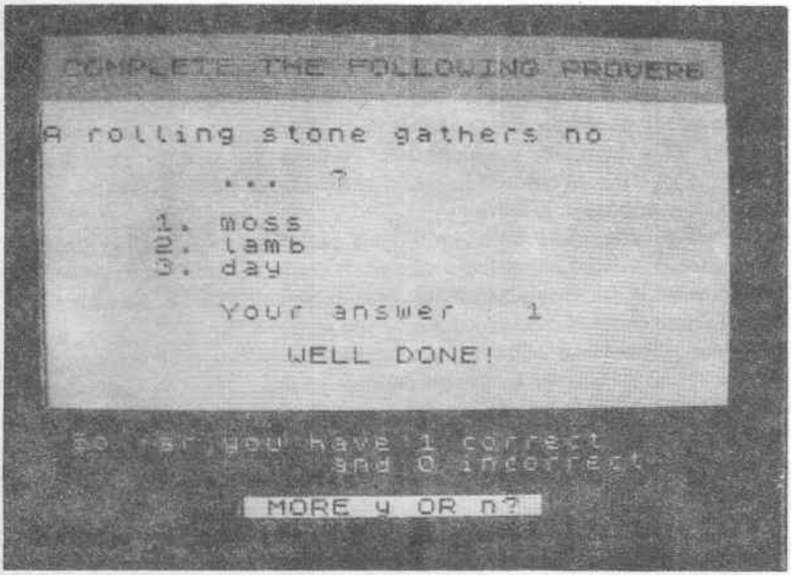
Interesujące są zestawy programów firmy LONGMAN przeznaczone do nauczania fizyki i chemii. Na przykład program *Dźwignie* wyjaśnia m.in. pojęcie momentu siły, a program *Obwody elektryczne* uczy prawa Ohma oraz tłumaczy działanie diody półprzewodnikowej i tranzystora. Wykład ilustrowany jest ruchomymi obrazami. Czasem są one dość dowcipne i wesołe — przyciągają więc uwagę ucznia. Programy przypominają trochę filmy oświatowe. Mają jednak nad nimi dużą przewagę. Angażują ucznia w proces nauczania. Opanowanie materiału ułatwiają bowiem zadania rozwiązywane początkowo przez komputer, który pokazuje tryb rozumowania, a następnie rozwiązywane przez ucznia pod kontrolą komputera.

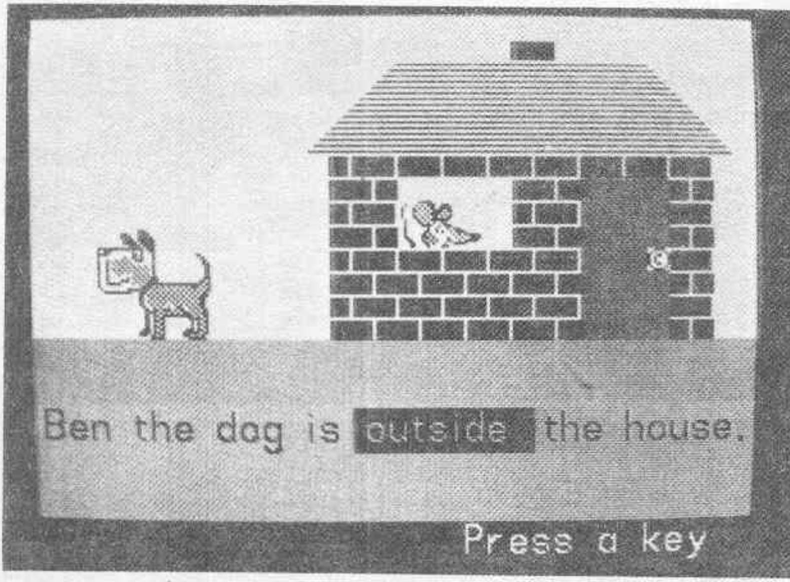
Napisano także wiele programów, które pomagają w nauce literatury angielskiej. Przykładem może być program firmy Penguin Study Software *Kupiec Wenecki* omawiający sztukę Szekspira. Zawiera on streszczenie poszczególnych scen. Można także zadawać komputerowi pytania dotyczące sztuki, np.: napisanie Shylock spowoduje wskazania w treści sztuki odpowiednich scen i dialogów związanych z tą postacią. Komputer następnie zadaje uczniowi tematy do opracowania. Podobnym programem jest *Antoniusz i Kleopatra* firmy Akadamias Software, *Hamlet*, *Henryk IV*, itd.

Kolejną dziedziną, w której korzysta się z programów komputerowych, jest nauczanie języków. Istnieją programy do nauki języka hiszpańskiego, niemieckiego, francuskiego i włoskiego. Do programów takich dołącza się często zwykłe kasety magnetofonowe, na których nagrane są dialogi i ćwiczenia fonetyczne. Programy komputerowe służą do ćwiczeń gramatyki. Zestawy firmy Rose Software *English 1* oraz *English 2* przeznaczone są do nauki języka angielskiego w szkołach angielskich dla uczniów 9—11-letnich. Mogą być w Polsce przystosowane do nauczania na poziomie średniozaawansowanym. Programy te uczą synonimów, przysłów, wyrazów o znaczeniach przeciwnych, ortografii, itd. W czasie ćwiczeń uczeń wybiera jeden z kilku wydrukowanych na ekranie wyrazów, ten który stanowi odpowiedni synonim wyrazu brakującego w zdaniu itd. (metoda ta nosi nazwę *Multiple choice*). Uczeń wskazuje słowo podając jedynie jego numer.

Najwięcej programów edukacyjnych jest przeznaczonych do nauczania początkowego — czytania oraz podstaw arytmetyki. Przykładem takich programów jest zestaw *Learn to Read* (nauka czytania) Firmy Macmillan Education. Początkowo dzieci uczy się słów. Nowe słowa wyświetlane na ekranie ilustrowane są rysunkami. W trzecim programie uczy się dzieci układania prostych zdań. Uczeń opisuje określone zdarzenie uzupełniając brakujące wyrazy w trzech kolejnych zdaniach. Pozostałe programy uczą kolejności liter alfabetu, zasad stosowania dużych liter oraz stosowania okoliczników miejsca (widocznie określanie miejsca sprawia dzieciom kłopoty i dlatego właśnie ten temat został wybrany). Nagrodą za dobre odpowiedzi jest zabawa. Czas zabawy zależy od liczby dobrych odpo-


Question 6
To burst.
The river its banks and flooded the village.
tense of the verb and
press ENTER.


COMPLETE THE FOLLOWING PROVERB
A rolling stone gathers no
... ?
1. moss
2. lamb
3. day
Your answer 1
WELL DONE!
So far you have 1 correct
and 0 incorrect
MORE y OR n?



wiedzi. Dla przykładu w programie *Capital Letters* (duże litery) zabawa polega na chwytaniu jabłek spadających z drzewa.

Wśród programów edukacyjnych poczesne miejsce zajmują gry. Gry takie bawiąc — uczą. Przykładem może być program *Survival* (Jak przeżyć?). Grający wybiera sobie zwierzę (np. mysz, orła lub nawet lwa). Każde zwierzę ma swych naturalnych wrogów, typowy pokarm, lepiej lub gorzej znosi niską temperaturę brak pożywienia i wody. Grający musi tak kierować swym zwierzątkiem, by jak najdłużej przeżyło w wyimaginowanym świecie. Dodatkowo, im dłużej dane zwierzę żyje, tym większe ma szanse na dalsze przetrwanie. Gra jest prosta i zawiera sporo ciekawostek o zwierzętach. Podobną grą jest *Sir Francis Drake* (firmy LCL). Grający podróżuje razem z bohaterem dookoła świata. Poznaje geografię oraz fakty z historii.

Są też programy dla osób, które chcą poszerzyć swoją wiedzę. Programy *How machine works* (Jak pracuje maszyna?) firmy Calpac Computer Software Inc opowiadają jak działa samochód oraz myśliwiec Harrier. Programy te są bogato ilustrowane ruchomymi, kolorowymi obrazkami.

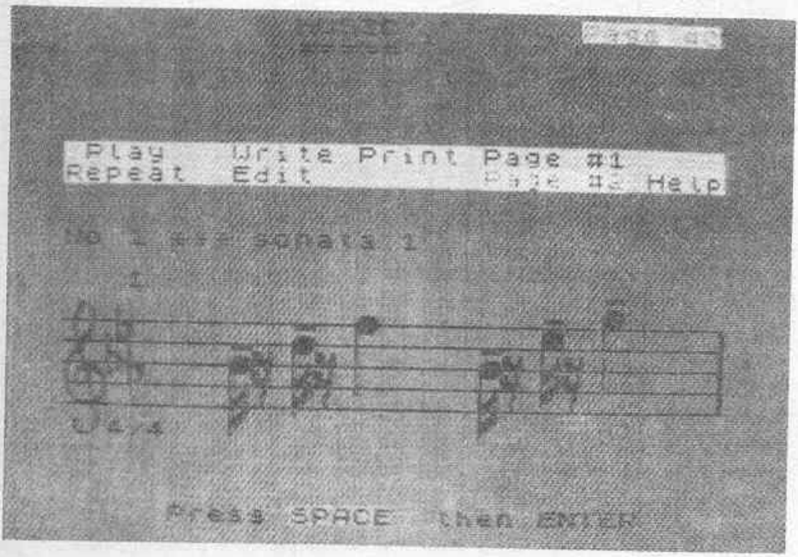
Firma Argus Press Software opracowała trzy zestawy z serii *Clever Clogs*: *Science* (Nauka), *General knowledge* (Wiedza ogólna), *The Arts* (Sztuka). Są to komputerowe quizy — każdy program zawiera pięć zestawów po 100 pytań. Komputer zadaje pytania z historii, przyrody, matematyki, fizyki, chemii i biologii, a program *The Art* z dziedziny poezji, muzyki, malarstwa, tańca, teatru i literatury angielskiej.

Kolejny program *Astronomer*, to marzenie wszystkich amatorów astronomii — domowe planetarium. Pozwala on oglądać na ekranie niebo widoczne z każdego

miejsca ziemi, o każdej porze roku. Program ten wyświetla także ruchomy model układu słonecznego.

Niektóre programy edukacyjne są wręcz zaskakujące. *Wheathermaster* (Macmillan Education) uczy naukowych podstaw przepowiadania pogody. Symuluje przy tym zmiany pogody nad Atlantykiem.

Duże zainteresowanie wzbudzają programy uczące muzyki. Wykorzystują one możliwości wytwarzania przez komputer dźwięku. Na przykład program *Music Master* uczy zasad zapisu nutowego, sposobu zapisywania nut, pauz, rytmu i melodii na pięciolinii. Uczeń ma możliwość wyboru jednego z kilku tematów, np. zasady podawania rytmu. Wykład wyświetlany na ekranie ilustrowany jest dźwiękami. Uczeń uczy się zależności między dźwiękiem i nutą. Może usłyszeć różnice między dźwiękami o różnej wysokości i różnym czasie trwania oraz zobaczyć, jak wygląda ich obraz zapisany na pięciolinii. Program daje także możliwość skomponowania własnej melodii, zapisania jej w pamięci RAM oraz na taśmie magnetofonowej, a następnie odegrania jej w dowolnym tempie. Autorzy zauważyli że program ten, przeznaczony dla dzieci, wzbudził zainteresowanie dorosłych. Pamiętali oni niezbyt dokładnie zasady zapisywania nut i chcieli dyskretnie uzupełnić swoją wiedzę o tym, na czym polega „cały ten jazz".



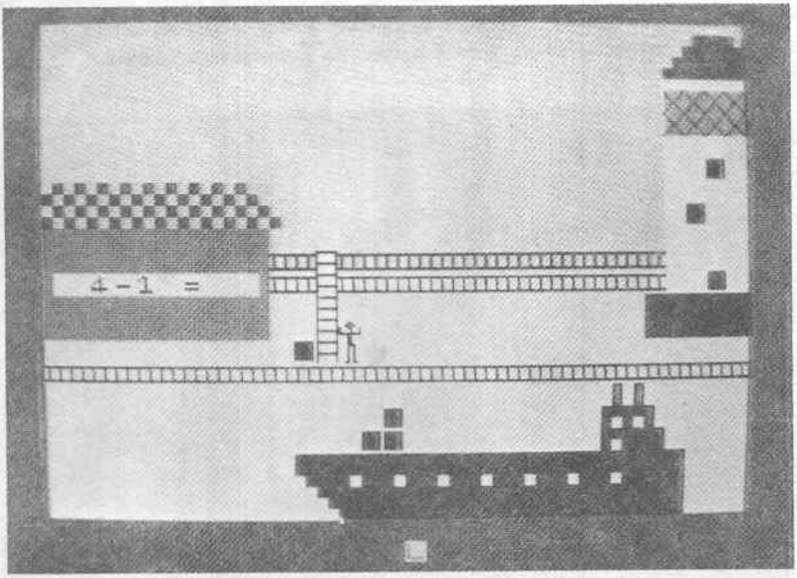
Jeśli mowa o dorosłych, to istnieją programy edukacyjne przeznaczone dla nich, także dla starszej młodzieży. Przykładem może być wymieniony już zestaw uczący języka Asemblera Z80 — *The Complete Machine Code Tutor* — lub *Beyond Basic*, albo *Go micro* (firmy LONGMAN) wyjaśniający sposób pracy i możliwości komputera.

Programy *Highway* (Przepisy drogowe) firm Datek Computing, Computer

Pentals i Rose Software wyjaśniają przepisy i znaki drogowe oraz sprawdzają ich znajomość. Mogą być pomocne osobom przygotowującym się do egzaminów na prawo jazdy lub na kartę rowerową.

Opracowano także programy edukacyjne dla nauczycieli. *Spectrum Graph-Plot* (firmy Calpac) przeznaczony jest dla nauczycieli matematyki i fizyki. Wykreśla on przebieg funkcji, na podstawie podanego zbioru punktów lub wzoru.

Jest wszakże pewien szkopuł. Programy te powstały w Anglii, więc wszystkie informacje wyświetlane na ekranie napisane są po angielsku. Tylko niektóre programy mogą być bez zmian używane przez osoby nie znające tego języka, np. programy do nauki języków oraz programy dla dzieci do nauki arytmetyki: dodawanie — odejmowanie, gdyż uczą one arytmetyki za pomocą zabawy — żadne informacje oprócz cyfr nie są drukowane na ekranie.



Dobry program edukacyjny musi być zaprojektowany przez doświadczonego pedagoga oraz specjalistę od psychologii uczenia się. Informatyk ma tu rolę wykonawcy.

Należy także ostrzec Czytelników, że większość istniejących programów to proste do zaprojektowania, lecz stosunkowo mało efektywne lekcje i ćwiczenia komputerowe. Firmy produkujące oprogramowanie często dają swym wyrobom etykietkę programów edukacyjnych, by je lepiej sprzedawać. Poza tym zastosowanie komputerów do wspomagania nauczania nie zawsze daje efekty proporcjonalne do nakładów, np. użycie komputera zamiast książki jest bezcelowe.

Niedostatki te nie zmniejszają jednak roli, jaka przypadnie komputerowemu wspomaganiu procesu nauczania.

* Gry 9.6

Cóż można napisać o grach? Powstały ich setki. Lepszych lub gorszych. Niektóre to prawdziwe majstersztyki programowania, pisane przez najlepszych programistów, z grafiką projektowaną przez plastyków i dźwiękiem tworzonym przez muzyków. Inne, to dzieła zapaleńców, np. *Ati Attack* wymyślony został przez 17--letniego chłopca.

Najbardziej fascynuje grafika gier, kolor i animacja. Szczególnie, jeśli zna się możliwości graficzne komputera. Czasem trudno uwierzyć, że ograniczonymi środkami jakie ma ZX Spectrum można było zrobić takie obrazy. Zachwyca także inwencja i poczucie humoru autorów gier.

Gry podzielić można na:

— zręcznościowe — przebieg gry zależy od zręczności w operowaniu klawiszami lub joystick'iem, np.: *Hungry Horace*, *Jumping Jack*, *PSST*, itd.,

— przygodowe — prezentujące pewną fabułę; grający utożsamia się z bohaterem i steruje jego ruchami, np.: *Ant Attack*,

— strategiczne — zarówno szachy (*Chess*) lub *Bridge*, jak i np. *Bitwa o Anglię*, czy *Dyktator*,

— symulacyjne — np. symulator lotu, *Flight Simulator*.

Wiele programów trudno zakwalifikować, bowiem łączą w sobie kilka cech. Podany podział należy potraktować więc jedynie orientacyjnie.

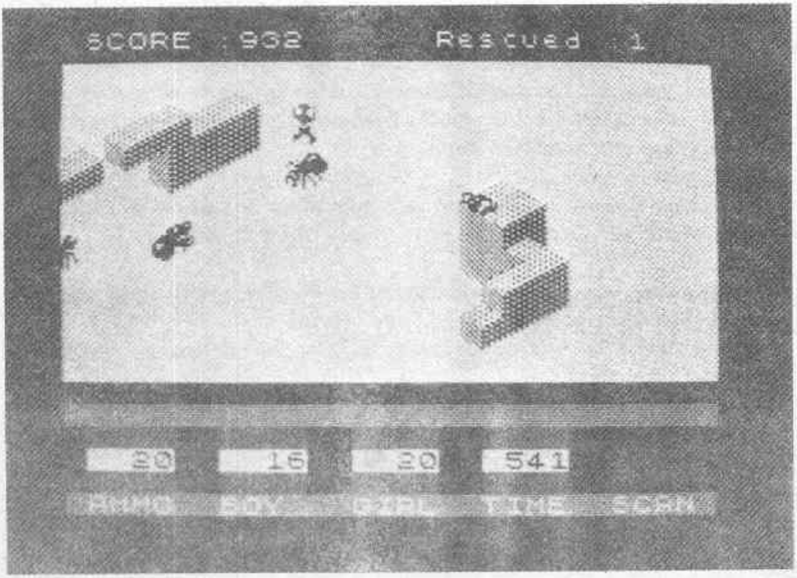
Gry sprzedawane są w Anglii w cenie od 5 do 15 funtów. W Polsce kopie tych programów kupić można np. z ogłoszenia za kilkadziesiąt lub co najwyżej kilkaset złotych. Inną metodą ich uzyskania jest wymiana. Często zapłatą za gry jest praca, np.: tłumaczenie instrukcji do programów. Czasem osoba, która sprowadza grę z zagranicy otrzymuje w zamian wiele programów rozpowszechnionych w kraju. W ten sposób docierają niekiedy do Polski nowości. Programy te kopiowane są następnie (bezprawnie) za pomocą specjalnych programów (np. *Copy-Copy*), które notabene są również sprzedawane w Anglii.

Trudno stwierdzić, które gry są najlepsze, najbardziej interesujące. Oceny i gusty ludzkie są bowiem bardzo różne. Jedni wolą gry zręcznościowe, inni fikcyjne przygody. Niełatwo pisać o nowościach, bowiem nowe gry wydają się znanymi od lat już po upływie dwu miesięcy. Opisane więc będą gry, które kiedyś Autorom wydały się interesujące lub wręcz zachwyciły.

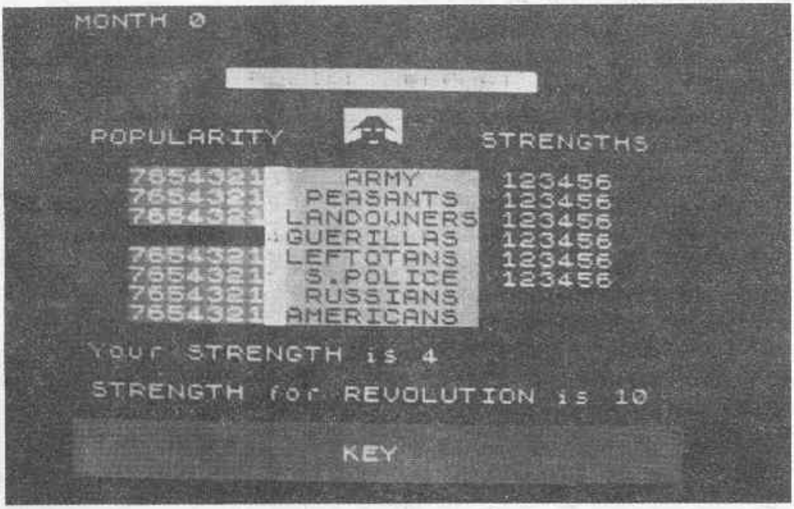
Jumping Jack (skaczący Jacuś) to najprostsza gra jaką można sobie wyobrazić. Na ekranie wyświetlane są poziome linie z przesuwającymi się przerwami. Narysowany kilkoma kreskami człowieczek biega wzdłuż owych linii i przeskakuje z poziomu na poziom przez dziury w liniach. Celem jest dotarcie na samą górę. Jeśli człowieczek upadnie traci na chwilę przytomność. Może wtedy spaść poziom

niżej. Spadek na samo dno kończy grę. Lakoniczny opis nie może oddać uroku gry. Zabawa jest naprawdę znakomita.

Ant Attack (Atak mrówek) jest grą przygodową. Bohater (chłopiec lub dziewczynka — grający wybiera według gustu) poszukuje partnera w mieście opanowanym przez krwiożercze mrówki. Grę tę warto polecić ze względu na grafikę. Na ekranie wyświetlony jest trójwymiarowy obraz murów i ograniczonego nim miasta. Widok oglądać można z dwu „kamer" przełączanych naciśnięciem klawisza. „Kamera" śledzi ruchy bohatera. Gdy ten porusza się, zmienia się perspektywa murów i budowli miasta. I to właśnie jest najbardziej fascynujące.



Dyktator to kolejna gra, przy której można łatwo stracić kilka godzin. Grający jest dyktatorem wymyślonego państwa znajdującego się gdzieś na Antypodach. Powinien jak najdłużej utrzymać się na tronie i zachować życie. Amatorów łatwego życia należy ostrzec. Rządy absolutne to ciężki kawałek chleba. Trzeba rządzić rozważnie, tak by zapewnić sobie współpracę tajnej policji, armii, właścicieli ziemskich, nie tracąc wpływów u chłopów. Należy zapobiegać rewolucji, trzymać w ryzach partyzantów, a w razie kłopotów finansowych dostać pożyczkę zagraniczną w USA lub ZSRR. Nie wolno zapomnieć o własnych potrzebach. Dyskretnie przelewać trzeba dolary na prywatne konto w Szwajcarii, a helikopter ratunkowy powinien być stale gotowy do startu. Czasem tajna policja odmawia współpracy, innym razem wybucha rewolucja lub następuje próba zamachu na życie głowy państwa. Czy takie doświadczenia nie zdołają zniechęcić do robienia kariery politycznej? (na Antypodach rzecz jasna).



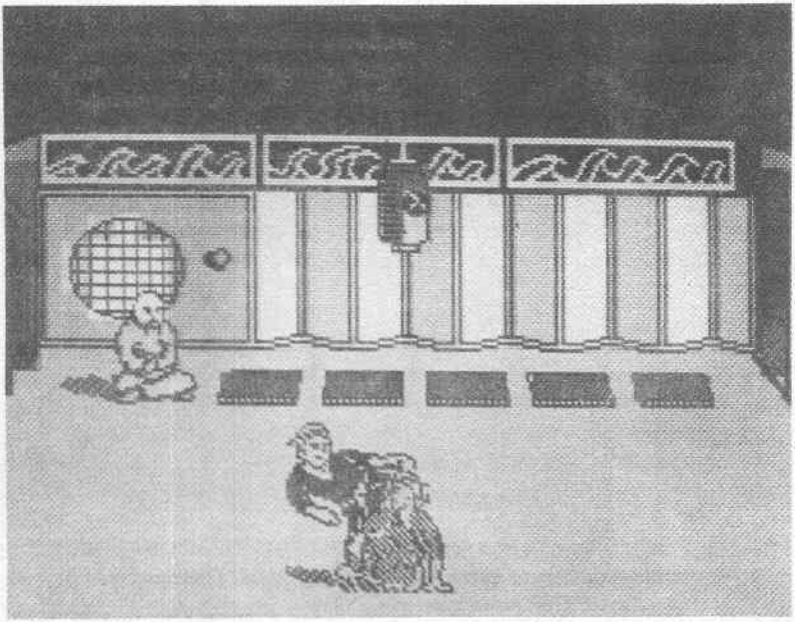
Konkurencja wśród producentów gier jest ogromna. Dlatego prześcigają się oni szukając pomysłów na nowe typy gry. Pojawiły się gry „karate", np.: *Bruce Lee*, *Kung Fu*, *Fist* (pięść). Ta ostatnia symuluje walkę karate między dwoma przeciwnikami. Uderzenia wybiera się wciskając odpowiednie klawisze. Obsługa gry wymaga używania 13 klawiszy — jest więc skomplikowana. Gra ta ma bardzo dobrą grafikę. Walczące postacie to dokładne rysunki ludzi w kimonach, a nie ludziki narysowane kilkoma kreskami. Postacie są znakomicie animowane, zmienia się nawet ich cień. W dali widać ośnieżony szczyt Fuji.

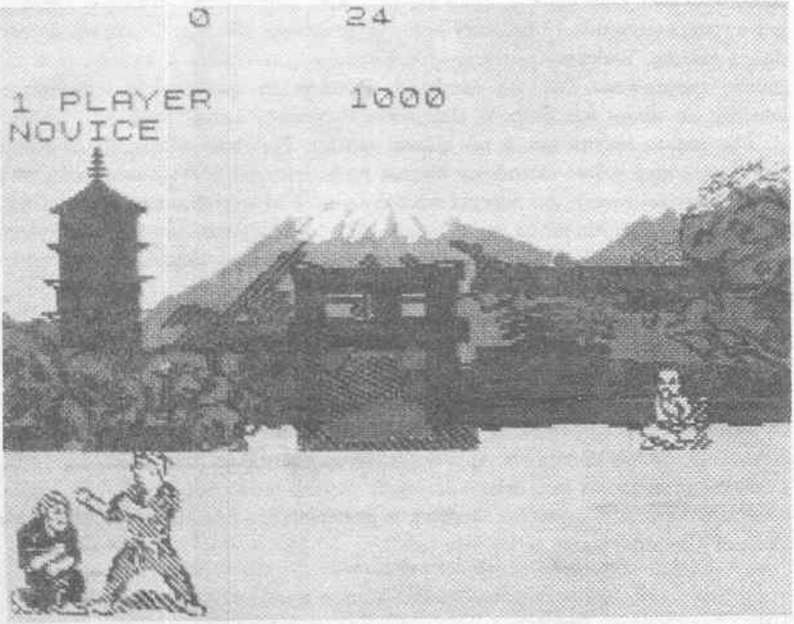
Nie jest to jedyna gra o tak dobrej grafice. Podobnie znakomitą animację ma tenis *Match point*: zawodnicy biegają po korcie, publiczność odwraca głowy za piłką, dzieci biegają po piłeczki wzdłuż siatki. I to wszystko na ZX Spectrum.

Po karate przyszła moda na szpiegów. Na rynku pojawiły się gry o przygodach agenta 007 — *James'a Bonda* oraz program *Spy contra spy* (Szpieg przeciw szpiegowi).

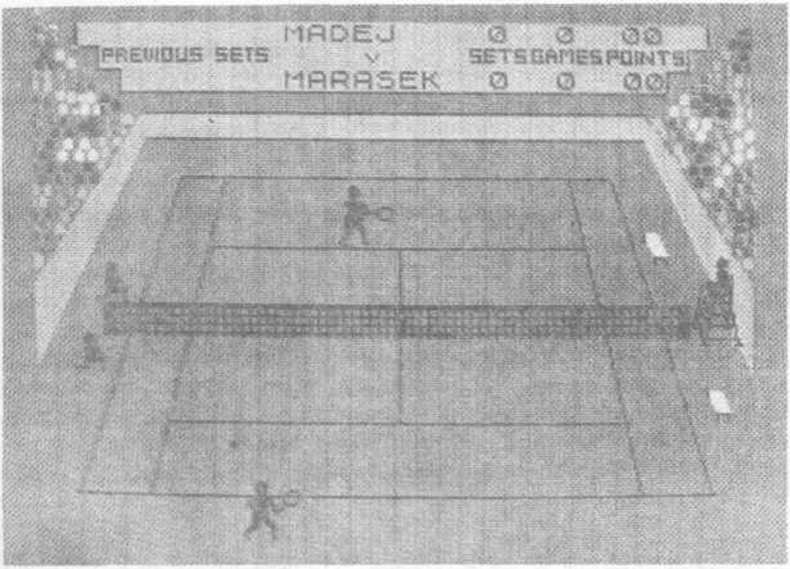
Kilka zdań poświęcić warto grom, które są rozrywką umysłową od wielu lat. Na początek gry karciane. Z „pokerów" najlepszy jest *Strippocker*. Dodatkowe utrudnienie polega na tym, że licytacja odbywa się po niemiecku. Nie zadowoli on jednak „zawodowych" pokerzystów podobnie, jak amatorów i znawców brydża nie usatysfakcjonują programy grające w tę grę. Dwa znane autorom programy: *Bridge* oraz *Bridge player 2* licytują w systemie Acol, nie najlepsza jest rozgrywka. Lepszym programem jest *Bridge player 2*, można zadawać ułożone lub losowe rozdania. Siła kart może być większa z jednej z par, komputer automatycznie „zrzuca" ostatnią kartę w kolorze itd.*).

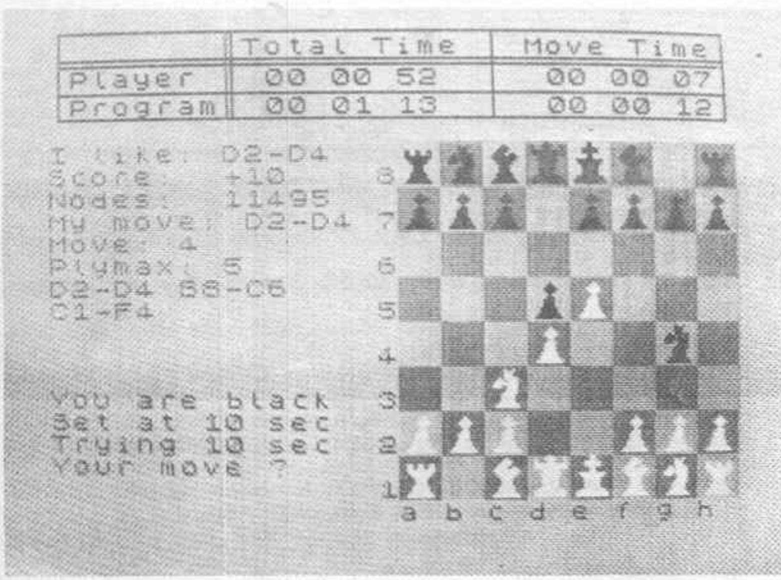
*) Być może „niedostatki" komputerowego brydża są spowodowane tym, że napisali go Anglicy, a nie polscy mistrzowie tej gry.





0
24
1 PLAYER
1000
NOVICE


MADEJ 0 0 00
PREVIOUS SETS
v
SETS GAMES POINTS
MARASEK 0 0 00


Total Time
Move Time
Player 00 00 52 00 00 07
Program 00 01 13 00 00 12
I like: D2-D4
Score: +10
Nodes: 11495
My move: D2-D4
Move: 4
Plymax: 5
D2-D4 B8-C6
C1-F4
You are black
Set at 10 sec
Trying 10 sec
Your move ?
8
7
6
5
4
3
2
1
a b c d e f g h

Szachy na Spectrum są zaskakująco dobre. Nawet wytrawni szachiści mogą być zadowoleni. Najprostszy program to *Chess* firmy Psion. Są także „mówiące szachy" — *Voice Chess*. Spectrum zapowiada (po angielsku) posunięcia, szacha i mata. Jeśli przewidzi ruch przeciwnika mówi: „I expected that" (spodziewałem się tego). Mowa jest niewyraźna, lecz zrozumiała. Najlepszym programem szachowym jest *Super chess 3*. Grający wybiera poziom od 1 do 9. Im wyższy poziom tym lepsza gra, lecz dłuższe oczekiwanie na ruch (np. na poziomie 7 — ok. 15 min.). Program ten sprawdzili szachiści w czasie przeglądu zorganizowanego jesienią 84 r. w klubie dziennikarza w Łodzi. Stwierdzili, że gra on na poziomie młodego zawodnika (nie amatora!). Czas oczekiwania na ruch (na wyższym poziomie) jest bardzo długi. Istnieje wersja tego programu (*Super chess 3.5*), napisana w całości w języku asemblera i działająca kilkanaście razy szybciej.

Żaden opis nie usatysfakcjonuje amatorów gier. Wolą zapewne grać niż czytać, a poza tym znają gry lepiej od autorów tego tekstu. Nie trzeba ich do grania zachęcać. Ludzi, dla których gry są na drugim planie, nie zamierzamy do takiej zabawy specjalnie namawiać — istnieją bowiem bardziej pożyteczne rozrywki. Mamy nadzieję, że zaprezentowane zdjęcia zastąpią zbędne opisy.

Literatura

1. Angell J. O., Jones B. J.: *Advanced Graphics with the Sinclair ZX Spectrum*. The Macmillan Press Ltd, 1983.
2. Clark K., McCabe F., Ennals J. R.: *ZX Spectrum micro Prolog. Primer*, LPA 1983.
3. Cooke S.: *The Companion to the Sinclair ZX Microdrive and Interfaces*. Pan Books Ltd, 1984.
4. Dickens A. C.: *Spectrum Hardware Manual*. Melbourne Hause Publishers, 1983.
5. Dickens A., Plumbey M., Whewell L.: *Spectrum Advanced User Guide*. Adder Publishers, 1984.
6. Dirksen A. J.: *Mikrokomputery*. WKŁ, Warszawa 1985.
7. Gascoigne S.: *Microchild learning through Logo*. Macmillan Publishers Ltd 1984.
8. Goldenberg E. P., Russell S. J., Cearter C. J. i inni: *Computers, Education And Special Needs*. Addison Wesley Publishing Company, 1984.
9. Hartnell T., Jones O.: *Programming your Spectrum*. Interface Publications Ltd, 1984.

10. Iglewski M., Madey J., Matwin S.: *Pascal*. WNT, Warszawa 1984.
11. James M.: *An expert guide to the Spectrum*. Granada Technical Books, Granada Ltd, 1984.
12. Johnson W.: *Fifteen Subroutines for the Sinclair Spectrum*. Sigma Technical Press, 1983.
13. Jones D.: *Beyond Simple Basic Delwing Deeper into your ZX Spectgum*. Interface Publications Ltd, 1983
14. Kluźniak T., Szpakowicz S.: *Prolog*. WNT, Warszawa 1983.
15. Lewenthal L. A., Sawille W.: *Z80 Assembly Language Subroutines*. Osborne, Mc Graw Hill, 1983.
16. Logan J.: *Understanding your Spectrum*. Melbourne Hause Publishers, 1982.
17. Logan J., O'Hara F.: *The complete Spectrum ROM disassembly*. Melbourne Hause Publishers, 1983.
18. McLean I., Gordon J.: *100 programs for the ZX Spectrum*. Prentice Hall International, 1984.
19. Merz J.: *Maschinencode Handbuch fur den ZX Spectrum*. Profisoft, 1983.
20. Misiurewicz P.: *Układy mikroprocesorowe*. WNT, Warszawa 1983.
21. Pieńkos J. i inni: *Modułowe systemy mikrokomputerowe*. WNT, Warszawa 1984.
22. Sinclair I.: *Introducing Spectrum Machine Code (How to get more speed and power)*. Granada Publishing, 1983.
23. Sparer E.: *Sinclair Logo 1 — Turtle Graphics*. Sinclair Research Ltd, 1984.
24. Sparer E.: *Sinclair Logo 2 — Programming Reference Manuel*. Sinclair Research Ltd, 1984.
25. Tang W.: *Spectrum Machine Code Language for the Absolute Beginner*. Melbourne Hause Publishers, 1982.
26. Thomossn D.: *Advanced Spectrum Forth*. Melbourne Hause, 1984.
27. Webb D.: *Super charge your Spectrum*. Melbourne Hause Publishers 1983.
28. Webb S.: *Practical Spectrum Machine Code Programming*. Virgin Books Ltd, 1984.
29. Winiewski J.: *Język C*. Informatyka 4, 5, 6/1985.
30. Wirth N.: *Wstęp do programowania systematycznego*. WNT, Warszawa 1978
31. Zaks R.: *Programming the Z80*. Melbourne Hause Publisher, 1981.
32. Zieliński R.: *Generatory liczb losowych. Programowanie i testowanie na maszynach cyfrowych*. WNT, Warszawa 1972.

W dodatku podano kody znaków oraz ich interpretację;
- Jako znaku lub instrukcji języka Basic,
- W kodzie heksadecymalnym,
- Jako instrukcję w języku asemblera:
— instrukcja jednobajtowa,
— instrukcja dwubajtowa (pierwszy bajt = CBH),
— instrukcja dwubajtowa (pierwszy bajt = EDH).

Kod	Znak	Heksa-decy-malnie	Instrukcja asemblera Z80		
			jednobajtowa	po CB	po ED
1	2	3	4	5	6
0	nie używane	00	nop	rlc b	
1		01	ld bc,NN	rlc c	
2		02	ld (bc),a	rlc d	
3		03	inc bc	rlc e	
4		04	inc b	rlc h	
5		05	dec b	rlc l	
6	PRINT przecinek*.	06	ld b,N	rlc (hl)	
7	EDIT	07	rlc a	rlc a	
8	kursor w lewo	08	ex af,af'	rrc b	
9	kursor w prawo	09	add hl,bc	rrc c	
10	kursor w dół	0A	ld a, (bc)	rrc d	
11	kursor w górę	0B	dec bc	rrc e	
12	DELETE	0C	inc c	rrc h	
13	ENTER	0D	dec c	rrc l	
14	nie używane	0E	ld c, N	rrc (hl)	
15		0F	rrc a	rrc a	
16	INK	10	djnz DIS	rl b	
17	PAPER	11	ld de, NN	rl c	
18	FLASH	12	ld (de), a	rl d	
19	BRIGHT	13	inc de	rl e	
20	INVERSE	14	inc d	rl h	
21	OVER	15	dec d	rl l	
22	AT	16	ld d, N	rl (hl)	
23	TAB	17	rla	rl a	
24	nie używane	18	jr DIS	rr b	
25		19	add hl, de	rr c	
26		1A	ld a, (de)	rr d	
27		1B	dec de	rr e	
28		1C	inc e	rr h	
29		1D	dec e	rr l	
30		1E	ld e,N	rr (hl)	
31		1F	rra	rr a	
32	spacja	20	jr nz,DIS	sla b	

*) Znaki o kodach 6÷23 są używane jako znki sterujące wyświetlaniem na ekranie.

1	2	3	4	5	6
33	!	21	ld hl,NN	sla c	
34	"	22	ld (NN),hl	sla d	
35	#	23	inc hl	sla e	
36	$	24	inc h	sla h	
37	%	25	dec h	sla l	
38	&	26	ld h,N	sla (hl)	
39	′	27	daa	sla a	
40	(	28	jr z,DIS	sra b	
41	)	29	add hl,hl	sra c	
42	*	2A	ld hl,(NN)	sra d	
43	+	2B	dec hl	sra e	
44	,	2C	inc l	sra h	
45	—	2D	dec l	sra l	
46	.	2E	ld l,N	sra (hl)	
47	/	2F	cpl	sra a	
48	0	30	jr nc,DIS		
49	1	31	ld sp,NN		
50	2	32	ld (NN),a		
51	3	33	inc sp		
52	4	34	inc (hl)		
53	5	35	dec (hl)		
54	6	36	ld (hl),N		
55	7	37	scf		
56	8	38	jr c,DIS	srl b	
57	9	39	add hl,sp	srl c	
58	:	3A	ld a,(NN)	srl d	
59	;	3B	dec sp	srl e	
60	<	3C	inc a	srl h	
61	=	3D	dec a	srl l	
62	>	3E	ld a,N	srl (hl)	
63	?	3F	ccf	srl a	
64	@	40	ld b,b	bit 0,b	in b, (c)
65	A	41	ld b,c	bit 0,c	out (c),b
66	B	42	ld b,d	bit 0,d	sbc hl,bc
67	C	43	ld b,e	bit 0,e	ld (NN), bc
68	D	44	ld b,h	bit 0,h	neg
69	E	45	ld b.I	bit 0,l	retn
70	F	46	ld b,(hl)	bit 0,(hl)	im 0
71	G	47	ld b.a	bit 0,a	ld i,a
72	H	48	ld c,b	bit 1,b	in c, (c)
73	I	49	ld c,c	bit 1,c	out (c),c
74	J	4A	ld c,d	bit 1,d	adc hl,bc
75	K	4B	ld c,e	bit 1,e	ld bc,(NN)
76	L	4C	ld c,h	bit 1,h	
77	M	4D	ld c,l	bit 1,l	reti
78	N	4E	ld c,(hl)	bit 1,(hl)	
79	O	4F	ld c,a	bit 1,a	ld r,a
80	P	50	ld d,b	bit 2,b	in d, (c)
81	Q	51	ld d,c	bit 2,c	out (c),d
82	R	52	ld d,d	bit 2,d	sbc hl,de
83	S	53	ld d,e	bit 2,e	ld (NN),de

1	2	3	4	5	6
84	T	54	ld d,h	bit 2,h	
85	U	55	ld d,l	bit 2,l	
86	V	56	ld d,(hl)	bit 2,(hl)	im 1
87	W	57	ld d,a	bit 2,a	ld a,i
88	X	58	ld e,b	bit 3,b	in e, (c)
89	Y	59	ld e,c	bit 3,c	out (c), e
90	Z	5A	ld e,d	bit 3,d	adc hl, de
91	[	5B	ld e,e	bit 3,e	ld de,(NN)
92	/	5C	ld e,h	bit 3,h	
93	]	5D	ld e,l	bit 3,l	
94	↑	5E	ld e,(hl)	bit 3,(hl)	im 2
95	_	5F	ld e,a	bit 3,a	ld a,r
96	£	60	ld h,b	bit 4,b	in h,(c)
97	a	61	ld h,c	bit 4,c	out (c),h
98	b	62	ld h,d	bit 4,d	sbc hl, hl
99	c	63	ld h,e	bit 4,e	ld (NN),hl
100	d	64	ld h,h	bit 4,h	
101	e	65	ld h,l	bit 4,l	
102	f	66	ld h,(hl)	bit 4,(hl)	
103	g	67	ld h,a	bit 4,a	rrd
104	h	68	ld l,b	bit 5,b	in l, (c)
105	i	69	ld l,c	bit 5,c	out (c), l
106	j	6A	ld l,d	bit 5,d	adc hl, hl
107	k	6B	ld l,e	bit 5,e	ld hl,(NN)
108	l	6C	ld l,h	bit 5,h	
109	m	6D	ld l,l	bit 5,l	
110	n	6E	ld l,(hl)	bit 5,(hl)	
111	o	6F	ld l,a	bit 5,a	rld
112	p	70	ld (hl),b	bit 6,b	in f,(c)
113	q	71	ld (hl),c	bit 6,c	
114	r	72	ld (hl),d	bit 6,d	sbc hl,sp
115	s	73	ld (hl),e	bit 6,e	ld (NN),sp
116	t	74	ld (hl),h	bit 6,h	
117	u	75	ld (hl),l	bit 6,l	
118	v	76	halt	bit 6,(hl)	
119	w	77	ld (hl),a	bit 6,a	
120	x	78	ld a,b	bit 7,b	in a, (c)
121	y	79	ld a,c	bit 7,c	out (c), a
122	z	7A	ld a,d	bit 7,d	adc hl,sp
123	{	7B	ld a,e	bit 7,e	ld sp,(NN).
124	|	7C	ld a,h	bit 7,h	
125	}	7D	ld a,l	bit 7,l	
126	—	7E	ld a,(hl)	bit 7,(hl)	
127	©	7F	ld a,a	bit 7,a	
128	□	80	add a,b	res 0,b	
129	▝	81	add a,c	res 0,c	
130	▘	82	add a,d	res 0,d	
131	▀	83	add a,e	res 0,e	
132	▗	84	add a,h	res 0,h	
133	▐	85	add a,l	res 0,l	
134	▚	86	add a,(hl)	res 0,(hl)	
135	▜	87	add a,a	res 0,a	
136	▖	88	adc a,b	res 1,b	

1	2	3	4	5	6
137	▞	89	adc a,c	res 1,c	
138	▌	8A	adc a,d	res 1,d	
139	▛	8B	adc a,e	res 1,e	
140	▄	8C	adc a,h	res 1,h	
141	▟	8D	adc a,l	res 1,l	
142	▙	8E	adc a,(hl)	res 1,(hl)	
143	█	8F	adc a,a	res 1,a	
144	(a)	90	sub b	res 2,b	
145	(b)	91	sub c	res 2,c	
146	(c)	92	sub d	res 2,d	
147	(d)	93	sub e	res 2,e	
148	(e)	94	sub h	res 2,h	
149	(f)	95	sub l	res 2,l	
150	(g)	96	sub (hl)	res 2,(hl)	
151	(h)	97	sub a	res 2,a	
152	(i)	98	sbc a,b	res 3,b	
153	(j)	99	sbc a,c	res 3,c	
154	(k) grafiki	9A	sbc a,d	res 3,d	
155	(l) definiowane	9B	sbc a,e	res 3,e	
156	(m)	9C	sbc a,h	res 3,h	
157	(n)	9D	sbc a,l	res 3,l	
158	(o)	9E	sbc a,(hl)	res 3,(hl)	
159	(p)	9F	sbc a,a	res 3,a	
160	(q)	A0	and b	res 4,b	ldi
161	(r)	A1	and c	res 4,c	cpi
162	(s)	A2	and d	res 4,d	ini
163	(t)	A3	and e	res 4,e	outi
164	(u)	A4	and h	res 4,h	
165	RND	A5	and l	res 4,l	
166	INKEY$	A6	and (hl)	res 4,(hl)	
167	PI	A7	and a	res 4,a	
168	FN	A8	xor b	res 5,b	ldd
169	POINT	A9	xor c	res 5,c	cpd
170	SCREEN$	AA	xor d	res 5,d	ind
171	ATTR	AB	xor e	res 5,e	outd
172	AT	AC	xor h	res 5,h	
173	TAB	AD	xor l	res 5,l	
174	VAL$	AE	xor (hl)	res 5,(hl)	
175	CODE	AF	xor a	res 5,a	
176	VAL	BO	or b	res 6,b	ldir
177	LEN	B1	or c	res 6,c	cdir
178	SIN	B2	or d	res 6,d	inir
179	COS	B3	or e	res 6,e	otir
180	TAN	B4	or h	res 6,h	
181	ASN	B5	or l	res 6,l	
182	ACS	B6	or (hl)	res 6,(hl)	
183	ATN	B7	or a	res 6,a	
184	LN	B8	cp b	res 7,b	lddr
185	EXP	B9	cp c	res 7,c	cpdr
186	INT	BA	cp d	res 7,d	indr
187	SQR	BB	cp e	res 7,e	otdr
188	SGN	BC	cp h	res 7,h	
189	ABS	BD	cp l	res 7,l	

1	2	3	4	5	6
190	PEEK	BE	cp (hl)	res 7,(hl)	
191	IN	BF	cp a	res 7,a	
192	USR	C0	ret nz	set 0,b	
193	STR$	C1	pop bc	set 0,c	
194	CHR$	C2	jp nz,NN	set 0,d	
195	NOT	C3	jp NN	set 0,e	
196	BIN	C4	call nz.NN	set 0,h	
197	OR	C5	push bc	set 0,l	
198	AND	C6	add a,N	set 0,(hl)	
199	<=	C7	rst 0	set 0,a	
200	>=	C8	ret z	set 1,b	
201	<>	C9	ret	set 1,c	
202	LINE	CA	jp z,NN	set 1,d	
203	THEN	CB		set 1,e	
204	TO	CC	call z,NN	set 1,h	
205	STEP	CD	call NN	set 1,l	
206	DEF FN	CE	adc a,N	set 1,(hl)	
207	CAT	CF	rst 8	set 1,a	
208	FORMAT	D0	ret nc	set 2,b	
209	MOVE	D1	pop de	set 2,c	
210	ERASE	D2	jp nc,NN	set 2,d	
211	OPEN #	D3	out (N),a	set 2,e	
212	CLOSE #	D4	call nc,NN	set 2,h	
213	MERGE	D5	push de	set 2,l	
214	VERIFY	D6	sub N	set 2,(hl)	
215	BEEP	D7	rst 16	set 2,a	
216	CIRCLE	D8	ret c	set 3,b	
217	INK	D9	exx	set 3,c	
218	PAPER	DA	jp c,NN	set 3,d	
219	FLASH	DB	in a,(N)	set 3,e	
220	BRIGHT	DC	call c,NN	set 3,h	
221	INVERSE	DD	poprzedza instrukcje dotyczące IX	set 3,l	
222	OVER	DE	sbc a,N	set 3,(hl)	
223	OUT	DF	rst 24	set 3,a	
224	LPRINT	E0	ret po	set 4,b	
225	LLIST	E1	pop hl	set 4,c	
226	STOP	E2	jp po,NN	set 4,d	
227	READ	E3	ex (sp),hl	set 4,e	
228	DATA	E4	call po,NN	set 4,h	
229	RESTORE	E5	push hl	set 4,l	
230	NEW	E6	and N	set 4, (hl)	
231	BORDER	E7	rst 32	set 4,a	
232	CONTINUE	E8	ret pe	set 5,b	
233	DIM	E9	jp (hl)	set 5,c	
234	REM	EA	jp pe,NN	set 5,d	
235	FOR	EB	ex de,hl	set 5,e	
236	GO TO	EC	call pe,NN	set 5,h	
237	GO SUB	ED		set 5,l	
238	INPUT	EE	xor N	set 5, (hl)	
239	LOAD	EF	rst 40	set 5,a	
240	LIST	F0	ret p	set 6,b	

1	2	3	4	5	6
241	LET	F1	pop af	set 6,c	
242	PAUSE	F2	jp p,NN	set 6,d	
243	NEXT	F3	di	set 6,e	
244	POKE	F4	call p,NN	set 6,h	
245	PRINT	F5	push af	set 6,l	
246	PLOT	F6	or N	set 6, (hl)	
247	RUN	F7	rst 48	set 6,a	
248	SAVE	F8	ret m	set 7,b	
249	RANDOMIZE	F9	ld sp,hl	set 7,c	
250	IF	FA	jp m,NN	set 7,d	
251	CLS	FB	ei	set 7,e	
252	DRAW	FC	call m,NN	set 7,h	
253	CLEAR	FD	poprzedza instrukcje dotyczące IY	set 7,l	
254	RETURN	FE	cp N	set 7, (hl)	
255	COPY	FF	rst 56	set 7,a	

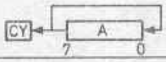
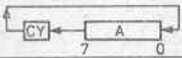
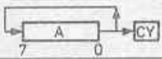
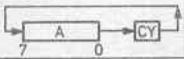
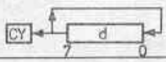
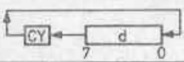
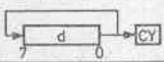
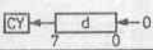
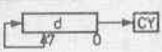
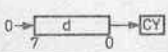
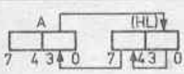
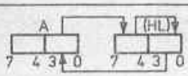
W dodatku tym podano w skrótowej formie listę instrukcji mikroprocesora Z80. Została ona podzielona na grupy w zależności od funkcji realizowanych przez kolejne instrukcje. Dla każdej z nich przedstawiono mnemonik, opis realizowanej czynności (zbliżony do zapisu w Basicu), stan znaczników, liczba bajtów zajmowanych przez kod instrukcji, czas wykonania w liczbie cykli zegara*), komentarz i wyjaśnienia (oznaczenia argumentów instrukcji). Kody instrukcji zamieszczono w dodatku A, łącznie z kodami ASCII i dodatkowymi kodami ZX Spectrum.

Przyjęto pewne stałe oznaczenia:

r — rejestry A,B,C,D,E,H,L,
rr′ — para rejestrów BC, DE, HL lub rejestr indeksowy IX, IY, SP,
A — akumulator,
n — liczba 8-bitowa,
nn — liczba 16-bitowa,
d — przesunięcie 8-bitowe (adres),
b — numer bitu 0,1,...7 (bit 0 najmniej znaczący),
(rr′) — zawartość miejsca pamięci adresowanego przez parę rejestrów (BC,DE,HL,IX+d, IY+d),
(nn) — zawartość miejsca pamięci o adresie nn,
IFF — znacznik maskowania przerwań,
I — rejestr przerwań,
R — rejestr odświeżania pamięci,
F — rejestr znaczników:
Z — zero,
C — przeniesienie,
PV — parzystość/namiar (w zależności od wykonywanej operacji),
S — znak,
N — znacznik odejmowania,
H — przeniesienie częściowe (do korekcji dziesiętnej)
↕ — znacznik zmieniany,
· — niezmieniany
X — nieokreślony
İ — operacja porównania.

*) Jeden takt zegara trwa ok. $3 \cdot 10^{-7}$ s w ZX Spectrum.

Tablica B.1. Instrukcje przesunięć

Mnemonik	Opis czynności	Znaczniki C	Z	P/U	S	N	H	Postać argumentów	T	L	Komentarz
RLCA		↕	•	•	•	0	0		4	1	
RLA		↕	•	•	•	0	0		4	1	
RRCA		↕	•	•	•	0	0		4	1	
RRA		↕	•	•	•	0	0		4	1	
RLC d		↕	↕	P	↕	0	0	d = r	8	2	d=r, (HL), (IX+d), (IY+d)
RL d		↕	↕	P	↕	0	0	d=(HL)	15	2	
RRC d		↕	↕	P	↕	0	0	d=(IX+d), (IY+d)	23	4	
RR d		↕	↕	P	↕	0	0				
SLA d		↕	↕	P	↕	0	0				
SRA d		↕	↕	P	↕	0	0				
SRL d		↕	↕	P	↕	0	0				
RLD		•	↕	P	↕	0	0		18	2	Przesunięcie cyfr w miejscu o adresie HL z wymianą bardziej znaczącej cyfry z mniej znaczącą cyfrą akumulatora; bardziej znacząca cyfra akumulatora zostaje niezmieniona
RRD		•	↕	P	↕	0	0		18	2	

Tablica B.2. Przesłania 8-bitowe

Mnemonik	Opis czynności	Znaczniki					
		C	Z	P/V	S	N	H
LD r,s	LET r = s	·	·	·	·	·	·
LD s,r	LET s = r	·	·	·	·	·	·
LD r,n	LET r = n	·	·	·	·	·	·
LD A,y	LET A = y	·	·	·	·	·	·
LD y,A	LET y = A	·	·	·	·	·	·
LD A,p	LET A = p	·	↕	İ	↕	0	0
LD p,A	LET p = A	·	·	·	·	·	·

İ — Zawartość przerzutnika blokady przerwań IFF kopiowana jest do znacznika P/V dla LDA,İ.

Tablica B.3. Przesłania 16-bitowe

Mnemonik	Opis czynności	Znaczniki					
		C	Z	P/V	S	N	H
LD rr′,nn	LET rr′ = nn	·	·	·	·	·	·
LD rr′,(nn)	LET rr′ = (nn)	·	·	·	·	·	·
LD (nn),rr′	LET (nn) = rr′	·	·	·	·	·	·
PUSH ss′	LET (SP) = s′ LET (SP−1) = s	·	·	·	·	·	·
POP ss′	LET s′ = (SP) LET s = (SP−1)	·	·	·	·	·	·
LD SP,yy′	LET SP = yy′	·	·	·	·	·	·

Tablica B.4. Wymiany rejestrów

Mnemonik	Opis czynności	Znaczniki					
		C	Z	P/V	S	N	H
EX DE, HL	DE ↔ HL	·	·	·	·	·	·
EX AF, AF′	AF ↔ AF′	·	·	·	·	·	·
EXX	[BC, DE, HL] ↔ [BC′, DE′, HL′]	·	·	·	·	·	·
EX (SP), rr′	(SP) ↔ rr′	·	·	·	·	·	·

Postać argumentów	Liczba taktów zegara	Liczba bajtów	Komentarz
s = r	4	1	r = A, B, C, D, D, E, H, L
s = (HL)	7	2	
s = (IX+d),(IY+d)	19	3	s = r,(HL),(IX+d),(IY+d)
	7	2	n = 0÷255
y = (BC),(DE)	7	1	y = (BC), (DE), (nn)
y = (nn)	13	3	
	9	2	p = I, R

Postać argumentów	Liczba taktów zegara	Liczba bajtów	Komentarz
rr′ = BC, DE, HL, SP	10	3	rr′ = BC, DE, HL, SP, IX, IY
rr′ = IX, IY	14	4	
rr′ = HL	16	3	
rr′ = BC,DE,SP,IX,IY	20	4	
ss′ = BC, DE, HL, AF	11	1	ss′ = BC, DE, HL, AF, IX, IY
ss′ = IX, IY	15	2	
ss′ = BC, DE, HL, AF	10	1	
ss′ = IX, IY	14	2	
yy′ = HL	6	1	yy′ = HL, IX, IY
yy′ = IX, IY	10	2	

Postać argumentów	Liczba taktów zegara	Liczba bajtów	Komentarz
	4	1	wymiana zawartości rejestrów DE i HL
	4	1	przełączenie się na parę rejestrów alternatywnych AF′
	4	1	przełączenie się na alternatywne BC′, DE′, HL′
rr′ = HL rr′ = IX, IY	19 23	1 2	rr′ = HL, IX, IY wymiana zawartości rejestrów rr′ ze szczytem stosu

Tablica B.5. Przesłania bloków i poszukiwania — operacje na ciągach słów

Mnemonik	Opis czynności	Znaczniki					
		C	Z	P/ /V	S	N	H
LDI	LET (DE) = (HL) LET DE = DE+1 LET HL = HL+1 LET BC = BC−1	·	·	①	·	0	0
LDD	LET (DE) = (HL) LET DE = DE−1 LET HL = HL−1 LET BC = BC−1	·	·	①	·	0	0
LDIR	10 LET (DE) = (HL): LET DE = DE+1: LET HL = HL+1: LET BC = BC−1 20 IF BC <> 0 THEN GOTO 10	·	·	0	·	0	0
LDDR	10 LET (DE) = (HL): LET DE = DE−1: LET HL = HL−1: LET BC = BC−1 20 IF BC <> 0 THEN GOTO 10	·	·	0	·	0	0
CPI	A | (HL) LET BC = BC−1 LET HL = HL+1	·	②	①	↕	1	↕
CPD	A | HL LET BC = BC−1 LET HL = HL−1	·	②	①	↕	1	↕
CPIR	10 A | (HL): LET BC = BC−1: LET HL = HL+1: 20 IF(BC <> 0) AND (A <>(HL)) THEN GOTO 10	·	②	①	↕	1	↕
CPDR	10 A | (HL): LET BC = BC−1 LET HL = HL−1 20 IF(BC <> 0) AND (A <> (HL) | THEN) GOTO 10	·	②	①	↕	1	↕

① Znacznik P/V = 0, jeżeli BC = 1, inaczej P/V = 1.
② Znacznik Z = 1, jeżeli A = (HL), inaczej Z = 0.

Warunki	Liczba taktów zegara	Liczba bajtów	Komentarz
	16	2	przesłanie bajtu z miejsca o adresie HL do miejsca o adresie DE z inkrementacją rejestrów DE i HL i dekrementacją rejestru BC
	16	2	jw. tylko z dekrementacją wszystkich par rejestrów (HL, DE, BC)
jeżeli BC = 0 jeżeli BC <> 0	16 21	2	powtarzanie LDI dopóki BC różne od zera
jeżeli BC = 0 jeżeli BC <> 0	16 21	2	powtarzanie LDD dopóki BC różne od zera
	16	2	porównanie A z zawartością pamięci o adresie HL, inkrementacja HL i dekrementacja BC
	16	2	jw. tylko z dekrementacją HL i BC
jeżeli BC = 0 lub A = (HL) jeżeli BC <> 0 i A <> (HL)	16 21	2	powtarzanie CPD dopóki BC różne od zera
jeżeli BC <> 0 i A <> (HL) jeżeli BC = 0 lub A = (HL)	21 16	2	powtarzanie CPD dopóki BC różne od zera

Tablica B.6. Arytmetyka i logika 8-bitowa

Mnemonik	Opis czynności	Znaczniki					
		C	Z	P/ /V	S	N	H
ADD s	LET A = A+s	↕	↕	V	↕	0	↕
ADC s	LET A = A+s+CY	↕	↕	V	↕	0	↕
SUB s	LET A = A−s	↕	↕	V	↕	1	↕
SBC s	LET A = A−s−CY	↕	↕	V	↕	1	↕
AND s	LET A = A AND s	0	↕	P	↕	0	↕
OR s	LET A = A OR s	0	↕	P	↕	0	↕
XOR s	LET A = A XOR s	0	↕	P	↕	0	↕
CP s	A|s	↕	↕	V	↕	1	↕
INC d	LET d = d+1	·	↕	V	↕	0	↕
DEC d	LET d = d−1	·	↕	V	↕	1	↕
DAA	$(A)_2 \rightarrow (A)_{10}$	↕	↕	P	↕	·	↕
CPL	LET A = NOT(A)	·	·	·	·	1	1
NEG	LET A = −A	↕	↕	V	↕	1	↕

Tablica B.7. Instrukcje specjalne i sterujące

Mnemonik	Opis czynności	Znaczniki					
		C	Z	P/ /V	S	N	H
NOP	nic nie rób	·	·	·	·	·	·
HALT	STOP	·	·	·	·	·	·
EI	włączenie przerwań (IFF = 0)	·	·	·	·	·	·
DI	zablokowanie przerwań (IFF = 1)	·	·	·	·	·	·
IM 0	zerowy tryb przyjmowania przerwań	·	·	·	·	·	·
IM 1	pierwszy tryb przyjmowania przerwań	·	·	·	·	·	·
IM 2	drugi tryb przyjmowania przerwań	·	·	·	·	·	·

Postać argumentów	Liczba taktów zegara	Liczba bajtów	Komentarz
s = r s = n s = (HL) s = (IX+d), (IY+d)	4 7 7 19	1 2 1 3	s = r, n, (HL), (IX+d), (IY+d) CY — zawartość znacznika C AND, OR, XOR — dotyczą każdego bitu argumentów XOR — suma modulo 2 porównanie A z s
d = r s = (HL) d = (IX+d), (IY+d)	4 11 23	1 1 3	d = r, (HL) (IX+d), (IY+d)
	4	1	korekcja dziesiętna A (zapis BCD)
	4	1	negacja wszystkich bitów A
	8	2	

Postać argumentów	Liczba taktów zegara	Liczba bajtów	Komentarz
	4	1	
	4	1	
	4	1	
	4	1	
	8	2	obsługa przerwań przez podprogramy RST
	8	2	obsługa przerwań przez RST 56
	8	2	obsługa przerwań przez podprogramy umieszczone w pamięci o adresie wskazywanym przez rejestr I (część bardziej znacząca) i urządzenia zewnętrzne

Tablica B.8. Arytmetyka 16-bitowa

Mnemonik	Opis czynności	Znaczniki					
		C	Z	P/V	S	N	H
ADD HL,ss'	LET HL = HL+ss'	$\updownarrow$	·	·	·	0	X
ADC HL,ss'	LET HL = HL+ss'+CY	$\updownarrow$	$\updownarrow$	V	$\updownarrow$	0	X
SBC HL,ss'	LET HL = HL−ss'−CY	$\updownarrow$	$\updownarrow$	V	$\updownarrow$	1	X
ADD IX,pp'	LET IX = IX+pp'	$\updownarrow$	·	·	·	0	X
ADD IY,rr'	LET IY = IY+rr'	$\updownarrow$	·	·	·	0	X
INC ss'	LET ss' = ss'+1	·	·	·	·	·	·
INC IX	LET IX = IX+1	·	·	·	·	·	·
INC IY	LET IY = IY+1	·	·	·	·	·	·
DEC ss'	LET ss' = ss'−1	·	·	·	·	·	·
DEC IX	LET IX = IX−1	·	·	·	·	·	·
DEC IY	LET IY = IY−1	·	·	·	·	·	·

Tablica B.9. Operacje na poszczególnych bitach

Mnemonik	Opis czynności	Znaczniki					
		C	Z	P/V	S	N	H
BIT b, s	LET Z = S_b	·	$\updownarrow$	X	X	0	1
SET b, s	LET S_b = 1	·	·	·	·	·	·
RES b, s	LET S_b = 0	·	·	·	·	·	·
CCF	LET CY = NOT(CY)	$\updownarrow$	·	·	·	0	X
SCF	LET CY = 1	1	·	·	·	0	0

Tablica B.10. Instrukcja skoków

Mnemonik	Opis czynności	Znaczniki					
		C	Z	P/V	S	N	H
JP nn	GOTO nn	·	·	·	·	·	·
JP cc, nn	IF cc THEN GOTO nn	·	·	·	·	·	·
JR e	GOTO e:REM PC = PC+e	·	·	·	·	·	·
JR w, e	IF w THEN GOTO e: REM PC = PC+e	·	·	·	·	·	·
JP (rr)	GOTO (rr)	·	·	·	·	·	·
DJNZ e	LET B = B−1 IF B<>0 THEN GOTO e	·	·	·	·	·	·

Postać argumentów	Liczba taktów zegara	Liczba bajtów	Komentarz
	11	1	ss' = BC, DE, HL,SP CY — zawartość znacznika C
	15	2	
	15	2	
	15	2	pp' = BC, DE, IX,SP
	15	2	rr' = BC, DE, IY,SP
	6	1	
	10	2	
	10	2	
	6	1	
	10	2	
	10	2	

Postać argumentów	Liczba taktów zegara	Liczba bajtów	Komentarz
s = r s = (HL) s = (IX+d), (IY+d)	8 12 20	2 2 4	s = r, (HL), (IX+d), (IY+d) znacznik Z przyjmuje wartość bitu o numerze b z komórki s
s = r s = (HL) s = (IX+d), (IY+d)	8 15 23	2 2 4	
	4	1	negacja znacznika CY
	4	1	ustawienie CY na 1

Warunek lub postać argumentów	Liczba taktów zegara	Liczba bajtów	Komentarz
	10	3	
	10	3	cc = NZ, Z, NC, C, PO, PE, P, M
	12	2	e = −128 ÷ +127
jeżeli warunek w spełniony jeżeli warunek w niespełniony	12 7	2	w = C, NC, Z, NZ
rr = HL rr = IX, IY	4 8	1 2	rr = HL, IX, IY
jeżeli B <> 0 jeżeli B = 0	13 8	2	

Tablica B.11. Operacje wejścia/wyjścia

Mnemonik	Opis czynności	Znaczniki C	Z	P/V	S	N	H
IN A, (n)	LET A = IN (n)	·	·	·	·	·	·
OUT (n), A	OUT n, A	·	·	·	·	·	·
IN r, (C)	LET r = IN(C)	·	↕	P	↕	0	↕
OUT (C), r	OUT C, r	·	·	·	·	·	·
INI	LET (HL) = IN(C) LET B = B−1 LET HL = HL+1	×	①	×	×	1	×
IND	LET (HL) = IN(C) LET B = B−1 LET HL = HL−1	×	①	×	×	1	×
OUTI	OUT C, (HL) LET B = B−1 LET HL = HL+1	×	①	×	×	1	×
OUTD	OUT C, (HL) LET B = B−1 LET HL = HL−1	×	①	×	×	1	×
INIR	10 LET (HL) = IN(C) LET B = B−1 LET HL = HL+1 20 IF B <> 0 THEN GOTO 10	×	1	×	×	1	×
INDR	10 LET (HL) = IN (C) LET B = B−1 LET HL = HL−1 20 IF B <> 0 THEN GOTO 10	×	1	×	×	1	×
OTIR	10 OUT C, (HL) LET B = B−1 LET HL = HL+1 20 IF B <> 0 THEN GOTO 10	×	1	×	×	1	×
OTDR	10 OUT C, (HL) LET B = B−1 LET HL = HL−1 20 IF B <> 0 THEN GOTO 10	×	1	×	×	1	×

① Z = 1 gdy B = 1, Z = 0 gdy B<>1

Warunek	Liczba taktów zegara	Liczba bajtów	Komentarz
	11	2	szyna adresowa A0÷A7 = n A8÷A15 = Akumulator
	12	2	szyna adresowa A0÷A7 = n A8÷A15 = Akumulator
	12	2	A0÷A7 = C A8÷A15 = B
	12	2	A0÷A7 = C A8÷A15 = B
	16	2	A0÷A7 = C A8÷A15 = B
	16	2	A0÷A7 = C A8÷A15 = B
	16	2	A0÷A7 = C A8÷A15 = B
	16	2	A0÷A7 = C A8÷A15 = B
jeżeli B <> 0 B = 0	21 16	2	powtarzanie INI dopóki B różne od zera
jeżeli B = 0 B <> 0	15 21	2	powtarzanie IND dopóki B różne od zera
			powtarzanie OUTI dopóki B różne od zera
			powtarzanie OUTD dopóki B różne od zera

Tablica B.12. Podprogramy

Mnemonik	Opis czynności	Znaczniki					
		C	Z	P/ /V	S	N	H
CALL nn	GOSUB nn	·	·	·	·	·	·
CALL cc, nn	IF cc THEN GOSUB nn	·	·	·	·	·	·
RET	RETURN	·	·	·	·	·	·
RET cc	IF cc THEN RETURN	·	·	·	·	·	·
RETI	RETURN z obsługi przerwania	·	·	·	·	·	·
RETN	RETURN z obsługi przerwania niemaskowalnego	·	·	·	·	·	·
RST p	GOSUB p	·	·	·	·	·	·

Wykaz zmiennych systemowych **Dodatek C**

W kolejnych kolumnach podano adres, długość (w bajtach), nazwę i zawartość zmiennych systemu ZX Spectrum. Nazwy zmiennych nie są rozpoznawane przez interpreter języka Basic i dostęp do nich możliwy jest jedynie za pomocą instrukcji POKE (zapis) i instrukcji PEEK (odczyt).

Adres	Liczba bajtów	Nazwa	Zawartość
1	2	3	4
23552	8	KSTATE	obszar roboczy procedury wczytania znaków z klawiatury
23560	1	LAST K	kod ostatnio wciśniętego klawisza
23561	1	REPDEL	czas (1/50s) przez jaki musi być wciśnięty klawisz, by został wczytany ponownie; początkowo 35
23562	1	REPPER	czas (1/50 s) pomiędzy kolejnymi wczytaniami wciśniętego klawisza; początkowo 5
23563	2	DEF ADD	adres argumentu funkcji FN
23565	1	K DATA	drugi bajt sterujący kolorem wprowadzonym z klawiatury
23566	2	TVDATA	bajty sterujące kolorem i pozycją (AT, TAB) znaków wprowadzanych na ekran

Postać argumentów	Liczba taktów zegara	Liczba bajtów	Komentarz
	17	3	
jeżeli warunek cc spełniony	17	3	cc = NZ, Z, NC, C, PO, PE, P, M
jeżeli warunek cc niespełniony	10		
	10	1	
jeżeli warunek cc spełniony	11	1	
jeżeli warunek cc niespełniony	5	1	
	14	2	
	14	2	
	11	1	p = 0, 8, 16, 24, 32, 40, 48, 56 wywołanie restartu (podprogramu o adresie p)

1	2	3	4
23568	38	STRMS	względne adresy kanałów dołączonych do poszczególnych strumieni: 23568 strumień 253 kanał „K" 23570 strumień 254 kanał „S" 23572 strumień 255 kanał „R" 23574 strumień 0 kanał „K" 23576 strumień 1 kanał „K" 23578 strumień 2 kanał „S" 23580 strumień 3 kanał „P" 23582 do 23606 strumienie 4÷15
23606	2	CHARS	adres (−256) graficznych form zbioru znaków (generatora znaków)
23608	1	RASP	długość dźwięku ostrzegawczego
23609	1	PIP	czas trwania dźwięku sygnalizującego wciśnięcie klawisza
23610	1	ERR NR	kod błędu minus 1; początkowo 255
23611	1	FLAGS	znaczniki wejścia/wyjścia bit: 0 — zapalony (= 1), jeżeli przed słowem kluczowym nie drukuje się spacji, 1 — zapalony, jeżeli współpracuje się z drukarką,

1	2	3	4
			2 — rodzaj kursora: zapalony — L/C, zgaszony (= 0) – K 3 — tryb wprowadzania: zapalony L/C, zgaszony K 5 — zapalony, jeżeli wciśnięto klawisz 6 — typ wprowadzonych danych: zapalony — liczba, zgaszony — łańcuch znaków 7 — zapalony — wykon. programu, zgaszony — sprawdzanie składni
23612	1	TV FLAG	znaczniki ekranu: 0 — zapalony, jeżeli wprowadzanie w dolną część ekranu (kanał „K") 3 — zapalony w trakcie wprowadzania linii poprawianej (tryb EDIT) 4 — zapalony przy automatycznym listowaniu 5 — zapalony, jeśli po wciśnięciu klawisza dolna część ekranu ma zostać wyzerowana
23613	2	ERR SP	adres procedury obsługi błędu
23615	2	LIST SP	adres powrotu dla automatycznego listowania
23617	1	MODE	rodzaj kursora: 0 dla L/C, 1 dla E, 2 dla G, 4 dla K
23618	2	NEWPPC	numer linii, do której wykonywany jest skok
23620	1	NSPPC	numer instrukcji w linii, do której wykonywany jest skok
23621	2	PPC	numer aktualnie wykonywanej linii
23623	4	SUBPPC	numer instrukcji w aktualnie wykonywanej linii
23624	1	BORDCR	kolor brzegu ekranu (*8)
23625	2	E PPC	numer linii, w której znajduje się kursor programu
23627	2	VARS	adres pola zmiennych programu w Basicu
23629	2	DEST	adres zmiennej, której aktualnie przypisywana jest wartość
23631	2	CHANS	adres pola rekordów kanałów
23633	2	CURCHL	adres aktualnie używanego kanału
23635	2	PROG	adres programu w języku Basic
23637	2	NXTLIN	adres następnej linii w programie
23639	2	DATADD	adres początku następnej danej w zbiorze DATA
23641	2	E LINE	adres aktualnie poprawianej instrukcji
23643	2	K CUR	miejsce kursora we wprowadzanej/poprawianej linii
23645	2	CH ADD	adres następnego znaku do interpretacji
23647	2	X PTR	adres znaku po znaczniku „?"
23649	2	WORKSP	adres obszaru roboczego interpretera
23651	2	STKBOT	adres początku stosu kalkulatora
23653	2	STKEND	adres końca stosu kalkulatora
23655	1	BREG	rejestr B kalkulatora

1	2	3	4
23656	2	MEM	adres pamięci kalkulatora (zwykle MEMBOT)
23658	1	FLAGS2	znaczniki: bit numer: 0 — zapalony, jeśli właśnie wyzerowano ekran, 1 — zgaszony, jeśli bufor drukarki jest pusty, 2 — zapalony, jeśli operuje się wewnątrz nawiasów, 3 — zapalony, jeśli CAPS LOCK wciśnięty (0), 4 — zapalony, jeśli ,,K" jest aktualnym kanałem
23659	1	DF SZ	liczba linii w dolnej części ekranu (oknie systemowym)
23660	2	S TOP	numer linii na początku automatycznego listingu
23662	2	OLDPPC	numer linii dla instrukcji CONTINUE
23664	1	OSPPC	numer instrukcji w linii dla CONTINUE
23665	1	FLAGX	znaczniki: bit numer: 1 — zapalony, jeśli było odwołanie do nieistniejącej zmiennej, 5 — zapalony — tryb INPUT, zgaszony — tryb EDIT, 7 — zapalony, jeśli wykonuje się INPUT LINE
23666	2	STRLEN	długość aktualnie analizowanego łańcucha znaków
23668	2	T ADDR	adres następnej pozycji w tabeli analizy syntaktycznej
23670	2	SEED	podstawa generacji liczb losowych (RND); ustawiana instrukcją RANDOMIZE
23672	3	FRAMES	licznik ramek TV zwiększany co 1/50 s; pierwszy bajt najmniej znaczący
23675	2	UDG	adres pola grafiki definiowanej przez użytkownika (UDG)
23677	2	COORDS	współrzędne x i y ostatnio rysowanego punktu (PLOT)
23679	1	P POSN	numer kolumny drukowania
23680	1	PR CC	mniej znaczący bajt adresu pozyji znaku do drukowania z bufora drukarki
23681	1	nie używany	
23682	2	ECHO E	numer kolumny i numer wiersza na ekranie, numer końca bufora dla wprowadzania (INPUT z kanału ,,K")
23684	2	DF CC	adres kursora w górnej części ekranu (kanał ,,S")
23686	2	DFCCL	adres kursora w dolnej części ekranu (kanał ,,K")
23688	2	SPOSN	zawiera 33 minus nr kolumny (pozycja dla PRINT w kanale ,,S") 24 munus nr wiersza

1	2	3	4
23690	2	SPOSNL	j.w. tylko dla dolnej części ekranu (kanał ,,K")
23692	1	SCR CT	liczba (+1) przesunięć linii na ekranie prowadzonych do pytania: scroll? (liczba linii, którą można wydrukować bez pytania: scroll?)
23693	1	ATTR P	globalne (dla całego ekranu) atrybuty
23694	1	MASK P	maska ,,przezroczystych" stałych kolorów
23695	1	ATTR T	lokalne atrybuty (dla aktualnie wykonywanej instrukcji PRINT, INPUT),
23696	1	MASK T	maska przeźroczystych atrybutów lokalnych,
23697	1	P FLAG	znaczniki barw (bity): 0 — wartość OVER (chwilowa), 1 — wartość OVER (stała), 2 — INVERSE (chwilowe), 3 — INVERSE (stałe), 4 — INK 9 (chwilowe), 5 — INK 9 (stałe), 6 — PAPER 9 (chwilowe), 7 — PAPER 9 (stałe)
23698	30	MEMBOT	pamięć kalkulatora
23728	2	nie używane	
23730	2	RAMTOP	adres ostatniego bajtu dostępnego dla Basica
23732	2	P RAMT	adres ostatniego bajtu dołączonej pamięci RAM
Zmienne układu Interface 1			
23734	1	FLAGS 3	znaczniki bit i zapalony podczas: 0 — realizacji rozszerzonego systemu instrukcji 1 — CLEAR, 2 — zmian ERR SP, 3 — pracy w sieci, 4 — LOAD, 5 — SAVE, 6 — MERGE, 7 — VERIFY
23735	2	VECTOR	adres używany do rozszerzania interpretera Basica (normalnie 01F0)
23737	10	SBRT	podprogram wywołania procedury z ROM-u Spectrum: LD HL, *wartość* CALL *adres* LD (23738), HL RET
23747	2	BAUD	szybkość przesyłania informacji w łączu RS 232 BAUD = (3500000/(26*szybkość))−2
23749	1	NTSTAT	własny numer stacji w sieci (1÷64)
23750	1	IOBORD	kolor brzegu ekranu podczas operacji wejścia/wyjścia
23751	2	SERFL	obszar roboczy RS232 — pierwszy bajt: znaczniki, drugi: wprowadzamy znak

1	2	3	4
23753	2	SECTOR	obszar roboczy microdrive'u — zwykle służy do zliczania sektorów
23755	2	CHAAD	chwilowe przechowywanie zawartości CHADD
23757	1	NTRESP	kod zwrotny stacji w sieci
23758	1	NTDEST	numer stacji przeznaczenia w sieci
23759	1	NTSRCE	numer stacji nadającej w sieci
23760	2	NTNUMB	liczba bloków informacji przesyłanych w sieci (0–65535)
23762	2	NTTYPE	nagłówek typu danych w sieci: 0 — dane, 1 — EOF
23763	1	NTLEN	długość bloku przesyłanego w sieci (0–255)
23764	1	NTDCS	suma kontrolna bloku przesyłanego w sieci
23765	1	NTHCS	suma kontrolna nagłówka bloku w sieci
23766	2	D STR 1	początek 8-bajtowego pola nagłówka pliku, drugi bajt — numer microdrive'u (1÷8) lub numer stacji przeznaczenia w sieci, lub szybkość pracy łącza szeregowego,
23768	1	S STR 1	numer strumienia (0÷15)
23769	1	L STR 11	typ urządzenia: „M", „N", „T", „B"
23770	2	N STR 1	długość nazwy pliku
23772	2	T STR 1	adres początku nazwy pliku
23774	8	D STR 2	kolejne 8 bajtów nagłówka pliku w instrukcji MOVE i LOAD
23782	1	HD 00	rodzaj pliku (zbioru) 0 — program, 1 — tablica liczb, 2 — tablica testów, 3 — ciąg bajtów,
23783	2	HD 0B	długość danych
23785	2	HD 0D	adres początku danych
23787	2	HD 0F	długość programu (lub nazwa tablicy)
23789	2	HD 11	numer linii do automatycznego startu programu
23791	1	COPIES	liczba kopii tworzonych instrukcją SAVE

Jeżeli w trakcie wykonywania programu lub ciągu instrukcji w trybie bezpośrednim nastąpi przerwanie pracy mikrokomputera, to w oknie systemowym ukazuje się odpowiedni komunikat. Informuje on o prawidłowym zakończeniu programu (lub poleceń w trybie bezpośrednim) lub o błędzie powstałym w czasie jego(ich) wykonywania. Każdy komunikat składa się z następującej sekwencji

— cyfry lub litery będącej kodem błędu,
— zwięzłej informacji o przyczynie zatrzymania,
— numeru linii, w której nastąpiło zatrzymanie pracy komputera,
— dwukropka,
— numeru instrukcji w linii, w której nastąpiło zatrzymanie pracy komputera.

Wszystkie instrukcje wykonywane w trybie bezpośrednim uznawane są za linię o numerze zero. Zgodnie z powyższym, komunikat

9 STOP Statement 100:3

oznacza, że zatrzymanie wykonania programu nastąpiło na skutek umieszczenia w linii 100 polecenia STOP jako trzeciej instrukcji tej linii.

ZX Spectrum ma „zapisane" w pamięci stałej następujące komunikaty:

Kod	Znaczenie	Możliwość wystąpienia
1	2	3
0	OK pomyślne zakończenie wykonania lub skok do nieistniejącej linii o numerze większym od każdej istniejącej.	zakończenie wykonania programu lub poleceń w trybie bezpośrednim
1	NEXT without FOR niezdefiniowana w instrukcji FOR zmienna sterująca pętlą (brak instrukcji FOR); niewykluczone jest istnienie zmiennej prostej o tej samej nazwie	NEXT
2	Variable not found użyto zmiennej uprzednio niezdefiniowanej w instrukcji LET, READ, INPUT, FOR, DIM lub niewczytanej z pamięci zewnętrznej	dowolna
3	Subscript wrong błędny indeks tablicy, większy od jej wymiaru lub niezgodna z definicją liczba indeksów tablicy	zmienna indeksowa lub łańcuch znaków
4	Out of memory zbyt mało pamięci potrzebnej interpreterowi do wykonania kolejnej instrukcji lub też próba wczytania programu o długości większej niż dostępna pamięć RAM; jeżeli jest to możliwe, to należy wsunąć część programu lub też zmienić wartość zmiennej RAMTOP na większą za pomocą instrukcji CLEAR *adres*	LET, DIM, INPUT, FOR GO SUB, LOAD, MERGE czasem przy obliczaniu wyrażeń

1	2	3
5	Out of screen próba drukowania na ekranie instrukcją PRINT AT w linii o nieistniejącym numerze (> 21) lub usiłowanie powiększenia okna systemowego instrukcją INPUT poza 23 wiersz ekranu	INPUT, PRINT, AT
6	Number too big pojawienie się w czasie obliczeń liczby większej od 10^{38}	dowolne wyrażenie arytmetyczne
7	RETURN without GO SUB wykonanie w programie instrukcji RETURN bez uprzedniego odwołania się do niej poprzez GO SUB	RETURN
8	End of file koniec pliku	microdrive
9	STOP statement wystąpienie w programie instrukcji STOP; wykonanie CONTINUE powoduje wykonanie następnej instrukcji programu	STOP
A	Invalid argument Niewłaściwy argument funkcji	SQR, LN, ASN, ACS, USR (z argumentem tekstowym) itd.,
B	Integer out of range liczba całkowita spoza dopuszczalnego zakresu dla danej instrukcji	RUN, RANDOMIZE POKE, DIM, GO TO, GO SUB, LIST, LLIST PAUSE, PLOT, CHR$, PEEK, USR (z argumentem liczbowym) tablice
C	Nonsense in Basic nieprawidłowy argument funkcji VAL, VAL$ bądź też uszkodzenie tekstu programu	VAL, VAL$
D	BREAK-CONTINUE repeats wciśnięcie klawisza BREAK w czasie wykonywania instrukcji wejścia/wyjścia; *CONTINUE powtarza podaną w spisie komunikatu instrukcję*	LOAD, SAVE, VERIFY MERGE, LLIST, LPRINT, COPY, gdy komputer zapyta: scroll?, a użytkownik wciśnie N lub BREAK/SPACE
E	Out of DATA przekroczenie, w czasie czytania instrukcji READ, zbioru utworzonego instrukcją DATA	READ
F	Invalid file name niewłaściwa nazwa pliku; występuje w przypadku instrukcji SAVE z nazwą pustą lub dłuższą niż 10 znaków	SAVE
G	No room for line brak wystarczającej ilości miejsca w pamięci na wprowadzenie nowej linii programu	wprowadzanie linii do programu

1	2	3
H	STOP in INPUT w trakcie wprowadzania danych za pomocą instrukcji INPUT jakieś dane wejściowe rozpoczęły się instrukcją STOP; w przypadku wprowadzania danych instrukcją INPUT LINE zostały przyciśnięte jednocześnie klawisze CAPS SHIFT oraz 6; *po wprowadzeniu CONTINUE instrukcja INPUT (czy INPUT LINE) zostaje powtórzona*	INPUT
I	FOR without NEXT błędna postać instrukcji FOR... TO... lub brak instrukcji NEXT	FOR
J	Invalid device stacja microdrive o podanym numerze nie jest podłączona	microdrive
K	Invalid color liczba określająca atrybut spoza zakresu	INK, PAPER, BORDER, FLASH, BRIGHT, OVER, INVERSE, a także po znakach kontrolnych (również w listingu)
L	BREAK into program przerwano wykonanie programu; w trakcie wykonywania programu został wciśnięty BREAK; numery linii i instrukcji w komunikacie odnoszą się do ostatnio wykonanej czynności, ale *CONTINUE powoduje przejście do następnej instrukcii* (ewentualne skoki są uwzględnione)	dowolna
M	RAMTOP no good niewłaściwie ustawiony RAMTOP — ostatni dostępny dla Bacica adres jest zbyt duży lub zbyt mały	CLEAR, możliwe w RUN, lub uszkodzenie komórki pamięci RAM.
N	Statement lost przejście do nieistniejącej instrukcji: uszkodzenie treści programu	RETURN, NEXT, CONTINUE etc.
O	Invalid stream błędne użycie strumienia danych	microdrive
P	FN without DEF FN próba wywołania instrukcją FN funkcji niezdefiniowanej za pomocą DEF FN	FN
Q	Parameter error błędny parametr funkcji; zła liczba argumentów w FN lub przynajmniej jeden z argumentów jest niewłaściwy (np.: tekstowy zamiast liczbowego i vice versa)	FN
R	Tape loading error błąd wczytania z taśmy	VERIFY, LOAD, MERGE

Poniższy program przeznaczony jest do modyfikowania i przeglądania zawartości pamięci ZX Spectrum. Za jego pomocą można wprowadzać do pamięci nie tylko dane, lecz także programy asemblerowe. Można je następnie uruchamiać zleceniem S.
Program jest na tyle prosty, że nie wymaga wyjaśnień. Wszystkie opcje wyszczególnione są w menu. Liczby można wprowadzać jako dziesiętne, bądź jeśli zakończenie znakiem H jako heksadecymalne.

```
  10 CLS : POKE 23658,8
 100 LET byte=900
 110 LET adres=2000
 120 LET menu=1000
 200 GO TO menu
 300 PRINT "  ";
 305 LET b=n
 310 IF d=1 THEN GO SUB 950
 320 IF d=0 THEN PRINT n
 330 RETURN
 900 REM byte
 910 LET b=PEEK l
 912 LET b1=b/16
 913 IF INT (b1)>9 THEN LET b1=b
1+7
 920 PRINT CHR$ (INT (b1)+48);
 922 LET b1=(b-(INT (b/16)*16))
 925 IF INT (b1)>9 THEN LET b1=b
1+7
 930 PRINT CHR$ (INT (b1)+48);
 940 RETURN
 950 REM hex adres
 951 LET h=4096
 952 LET b1=b/h
 953 IF INT (b1)>9 THEN LET b1=b
1+7
 955 PRINT CHR$ (INT (b1)+48);
 960 LET b=b-INT (b/h)*h
 965 LET h=h/16
 970 IF h>=1 THEN GO TO 952
 975 PRINT "h";TAB 7;
 980 RETURN
 990 CLS
 992 LET i=0
 999 RETURN
1010 PRINT TAB 7;"MINI-MONITOR";
''
1030 PRINT TAB 7;"Z) zmienianie"
1035 PRINT
1040 PRINT TAB 7;"O) ogladanie"
1045 PRINT
1050 PRINT TAB 7;"S) start progr
amu"
```

```
1055 PRINT
1060 PRINT TAB 7;"K) koniec"
1080 IF INKEY$="" THEN GO TO 108
0
1090 IF INKEY$="Z" THEN GO SUB 2
500
1100 IF INKEY$="O" THEN GO SUB 4
600
1110 IF INKEY$="S" THEN GO SUB 3
000
1120 IF INKEY$="K" THEN GO SUB 3
900
1125 CLS
1130 GO TO menu
2000 REM adres
2005 LET d=1
2010 CLS
2020 PRINT "adres: "
2030 INPUT a$
2040 PRINT a$
2045 IF (CODE (a$(1 TO 1))>73) O
R (LEN (a$)>5) THEN GO TO 2000
2050 IF a$(LEN a$)="H" THEN LET
d=0: GO TO 2100
2060 LET n=VAL a$
2065 REM PRINT n
2070 RETURN
2100 LET n=0
2110 FOR x=1 TO LEN a$-1
2111 LET b1=CODE a$(x): IF b1>96
 THEN LET b1=b1-32
2112 LET b1=b1-48
2113 IF b1>9 THEN LET b1=b1-7
2120 LET n=n*16+b1
2130 NEXT x
2140 RETURN
2500 REM zmienianie
2505 LET i=0
2510 CLS
2520 GO SUB adres
2521 GO SUB 300
2522 LET l=n
2525 PRINT AT 21,0;"X)stop ENTER
)do przodu R)wstecz"
2527 PRINT AT 2,0;
2530 IF d=1 THEN PRINT l;TAB 7;
2532 LET b=l

2535 IF d=0 THEN GO SUB 950
2540 GO SUB byte
2550 PRINT "-->";
2560 INPUT a$
2570 IF a$<>"" THEN GO TO 2600
2580 LET l=l+1
2585 PRINT
2587 LET i=i+1
2590 IF i>18 THEN GO SUB 990
2595 GO TO 2530
2600 IF (LEN a$>3) THEN GO TO 25
60
```

```
2610 IF a$="R" THEN GO TO 2700
2620 IF a$="X" THEN RETURN
2625 IF CODE (a$(1 TO 1))>73 THE
N GO TO 2560
2630 GO SUB 2050
2635 LET b=n
2640 GO SUB 912
2650 POKE l,n
2660 GO TO 2580
2700 LET l=l-1
2710 GO TO 2585
3010 GO SUB adres
3015 GO SUB 300
3020 RANDOMIZE USR n
3030 PAUSE 4e4
3040 RETURN
3900 CLS
3910 STOP
4610 GO SUB adres
4612 GO SUB 300
4615 PRINT AT 20,0;" DOWOLNY) da
lej M)menu"
4618 PRINT AT 2,0
4620 FOR s=0 TO 15
4630 IF d=1 THEN PRINT n+s*8;TAB
 7;
4632 LET b=n+s*8
4635 IF d=0 THEN GO SUB 950
4640 FOR x=0 TO 7
4650 LET l=n+s*8+x
4660 GO SUB byte
4670 PRINT " ";
4680 NEXT x
4690 PRINT
4692 IF INKEY$="" THEN GO TO 469
2
4695 IF INKEY$="M" THEN GO TO 47
60
4700 NEXT s
4710 PRINT
4740 LET n=l+1
4745 CLS
4750 GO TO 4615
4760 RETURN
5000 LET b=20480
5010 GO SUB 950
```

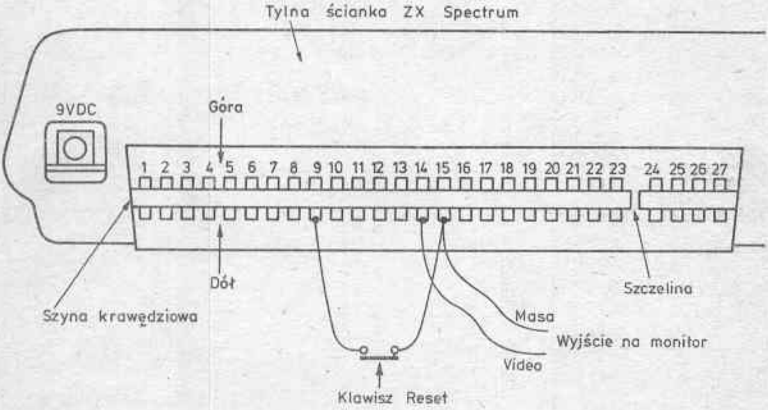
Szyna krawędziowa — Dodatek F

Wszystkie sygnały używane w systemie ZX Spectrum wyprowadzone są na zewnątrz za pomocą tzw. szyny krawędziowej (ang. *edge connector*). Na szynie tej znajdują się następujące sygnały:

— szyna danych (DO÷D7),
— szyna adresowa (A0÷A15),

— wszystkie sygnały sterujące procesora Z80 (BUSACK*⁾, RFSH, M1, WAIT, RESET, WR, BUSRQ, RD, IORQ, MREQ, HALT, NMI, INT, IORQE)
— napięcia (0 V, −12 V, +12 V, −5 V, +5 V, +9 V),
— zegar (CK),
— linia odłączająca ROM (ROMCS),
— wyjścia modulatora VHF: VIDEO — całkowity sygnał wizyjny;
Y — sygnał luminancji i synchronizacji,
V — sygnał różnicowy: czerwony — Y,
U — sygnał różnicowy: niebieski — Y.

*⁾ Kreska nad nazwą oznacza, że aktywny jest niski poziom sygnału.



Tablica E. Sygnały na szynie krawędziowej

Nr	Góra	Dół	Nr	Góra	Dół
1	wolne	A11	15	NMI	OV
2	A10	A9	16	INT	IORQE
3	A8	BUSACK	17	D4	A3
4	RFSH	ROMCS	18	D3	A2
5	M1	A4	19	D5	A1
6	−12 V	A5	20	D6	A0
7	+12 V	A6	21	D2	CK
8	WAIT	A7	22	D1	0V
9	−5 V	RESET	23	D0	0V
10	WR	BUSRQ	24	wolne	+9 V
11	RD	U	25	D7	+5 V
12	IORQ	V	26	A13	A12
13	MREQ	Y	27	A15	A14
14	HALT	VIDEO			

Cena zł 490.–

ISBN 83-206-0691-8